

GWD-R.72
SAGA-CORE-WG

Tom Goodale, Cardiff University
Shantenu Jha, University College London
Thilo Kielmann, Vrije Universiteit, Amsterdam
Andre Merzky, Vrije Universiteit, Amsterdam
John Shalf, Lawrence Berkeley National Laboratory
Christopher Smith, Platform Computing

Version: 1.0 RC.4

October 12, 2006

A Simple API for Grid Applications (SAGA)

Status of This Document

This document provides information to the grid community, proposing a standard for a simple API for grid applications. 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 (2006). All Rights Reserved.

Abstract

This document specifies the Simple API for Grid Applications (SAGA), 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").

Contents

1	Introduction	4
1.1	How to read this Document	4
1.2	Notational Conventions	4
1.3	Security Considerations	5

2	General Design Considerations	6
2.1	API Scope and Design Process	6
2.2	The SIDL Interface Definition Language	10
2.3	Language Binding Issues	14
2.4	Compliant Implementations	15
2.5	Object Management	17
2.6	Asynchronous Operations and Concurrency	21
2.7	State Diagrams	23
2.8	Execution Semantics and Consistency Model	23
2.9	Optimizing Implementations, Latency Hiding	25
2.10	Configuration Management	25
2.11	The 'URL Problem'	26
2.12	Miscellaneous Issues	28
3	SAGA API Specification	29
	Look & Feel API Packages	29
3.1	SAGA Error Handling	31
3.2	SAGA Base Object	41
3.3	SAGA Session Handling	46
3.4	SAGA Context	51
3.5	SAGA Attribute Interface	56
3.6	SAGA Monitoring Model	68
3.7	SAGA Task Model	93
	Functional API Packages	109
3.8	SAGA Job Management	110

3.9 SAGA Name Spaces	139
3.10 SAGA File Management	178
3.11 SAGA Replica Management	197
3.12 SAGA Streams	209
3.13 SAGA Remote Procedure Call	228
4 Intellectual Property Issues	237
4.1 Contributors	237
4.2 Intellectual Property Statement	238
4.3 Disclaimer	238
4.4 Full Copyright Notice	238
A SAGA Code Examples	240
References	248

1 Introduction

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

1. Provide a **simple** API that can be used with much less effort compared to the vanilla interfaces of existing grid middleware. A guiding principle for achieving this simplicity is the *80-20 rule*: serve 80% of the use cases with 20% of the effort needed for serving 100% of all possible requirements.
2. Provide a standardized, common interface across various grid middleware systems and their versions.

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.

An implementor of the SAGA API should read the complete document carefully. It will very likely be insufficient to extract the embedded SIDL specification of the API, and hope to 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 SAGA compliant.

This document is structured as follows. This Section focusses on the formal aspects on an OGF recommendation document. Section 2 outlines the general design considerations of the SAGA API. Section 3 contains the SAGA API specification itself. Section 4 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 [4].

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 3.4 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 those aspects of the SAGA API specification that are applicable to most or all of the SAGA packages as defined in Section 3.

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 [12]. From these use cases, the requirements on a SAGA API have been derived. The requirements analysis has been published as GFD.71 [13]. For the actual API definition (this document), the *SAGA-CORE Working Group* (SAGA-CORE-WG) has been established.

2.1.1 Requirements from the SAGA Requirement Analysis

The SAGA Requirement Analysis [13] lists the following, functional and non-functional requirements on 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.
- Asynchronous notification 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.

- 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 error support 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 API level.

2.1.2 Requirement Adoption Strategy

The use cases expressed the above requirements with 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 no use cases from the enterprise user community, and also had no active participation from that community in the SAGA standardization process, no enterprise specific API package is included here. This does not imply that we consider them unnecessary, but rather reflects our wish to orient the API on real use cases, and to avoid the creation of an API for made up use cases, and from half-baked expertise.

Scope of the SAGA API

As various sides expressed their need for the availability 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 requirements is deferred to future versions, or extensions, of the SAGA API. Based upon this reasoning, areas of functionality (from now referred to as *packages*) that are included in SAGA API are the following:

- jobs
- files (and logical files)
- streams
- auxiliary API's for
- GridRPC [14]
 - 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 [1])
- GridCPR [7]
- 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 API's, which is considered orthogonal to the non-auxiliary SAGA packages.

Dependencies between packages have been kept to a minimal level, to allow each package to be used independently of any other; this also may 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. 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 [6]. This is NOT correct. Reasons are the following:

- OGSA applies to service and middleware level.
SAGA applies to 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 the mostly used grid functionalities on 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 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 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 seamless 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*, [3] for specifying the API. This provides a programming-language neutral representation 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 of multi-dimensional arrays, etc. Although the SAGA spec does not use these features extensively, the multi language 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 CONST

- **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** supposed to show up in the implementations 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 multidimensional 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 to 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 3.7.

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` on the 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 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*. The format definitions for these specifications can be found in section 3.5 "SAGA Attribute Interface" and section 3.6 "SAGA Monitoring Model", respectively.

2.2.5 Method Specification Details

All methods defined in the SIDL specification sections are further explained in the 'Details' sections in this document. These details to method specifications are *normative*. They are formatted as follows (example taken from the `saga::ns_directory` class:

```
- move
  Purpose: rename source to target, or move source to
           target if target is a directory.
  Format:  move           (in string   source,
                          in string   target,
                          in int      flags);
  Inputs: source:       name to move
```

```

        target:          name to move to
        flags:           flags defining the operation
                        modus
Outputs: -
Throws:  BadParameter
        DoesNotExist
        IncorrectState
        AlreadyExists
Notes:  - if the target already exists, it will be
        overwritten if the 'Overwrite' flag is set,
        otherwise it an 'AlreadyExists' exception is
        thrown
        - moving '.' is not allowed, and throws
        a 'BadParameter' exception
        - default flag set is 'None' (0)
        - similar to 'mv' as defined 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
Outputs:   descriptions of out parameters
InOuts:   descriptions of inout parameters
Throws:    list of exceptions the method can throw
PreCond:   conditions for successful invocation
PostCond:  effects of successful invocation
Notes:    other details

```

PreCond'itions are often left out if there are none. An example for a precondition is a specific object state.

PostCondtions are often left out, if these are deemed sufficiently covered in the **Purpose** part. An example for a postcondition is a changed object state.

Exceptions listed under **Throws** are the only ones which can be thrown by the method.

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

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 interfaced holds: if methods are overloaded (i.e. redefined with the same name), the semantics of the overloaded methods still applies (i.e. all Notes given on the detailed method description apply). That does also hold for **CONSTRUCTORS** and **DESTRUCTORS**, and also for example for a `close()` which si implecately called on the base class' destruction.

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-and-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-and-feel of the respective programming language. This comprises the syntax for the entities (objects, methods, classes, etc.), but also, to some degree, semantical details for which it makes sense to vary them with the programming language. We summarize the semantic-ddetails 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 suitable 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, 64bit 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"
```

- Object life time 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. This means that any method MUST be implemented with the syntax and with the semantics specified in this document, 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.

A 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.

A 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 (`exception`, `error_handler`, `object`, `session`, `context`, `attribute`, `callback`, `metric`, `monitorable`, `steerable`, `async`, `task`, and `task_container`) MUST be implemented completely for an implementation to be compliant. A partial 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 a declared goal of the SAGA approach.

The SAGA CORE Working Group will strive to provide, along with the language binding documents, compliance 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 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 Section 3 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. More important, objects and methods from different sessions are guaranteed to *not* to share any information, and are completely shielded from each other. This will allow 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 parents 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 those 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 of 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 original 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 changes not, it is copied
```

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 (see Section 3.1)
- callbacks on metrics (see Section 3.6)

Not copying previous error conditions disambiguates error handling. 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 merely 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.

2.5.3 Object State Life Time

In general, the life time of SAGA object instances is defined as natively expected in the respective languages, so 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 life time of the context in which they have been 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.

Due to the diversity of life time 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:

- a 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 that 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 immediately¹.
2. If the immediate de-allocation of resources is not possible, for whichever reasons, the 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. If resource de-allocation does not succeed within this timeout period, a `NoSuccess` exception **MUST** be thrown. Negative values imply to wait forever, a value of zero (the default) implies that the method can return immediately, 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.

2.6 Asynchronous Operations and Concurrency

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

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

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

method call. The SIDL specification lists only the synchronous version of the API methods, but all packages implementing the task interface **MUST** provide the various asynchronous methods as well. Please see section 3.7 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 that parameters is *always* as follows:

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

These methods do *not* cause a `TimeOut` exception as the timeout period passes, but return silently. For an 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 deallocation 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 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 allows state transitions. These diagrams are *normative*.

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 ap-

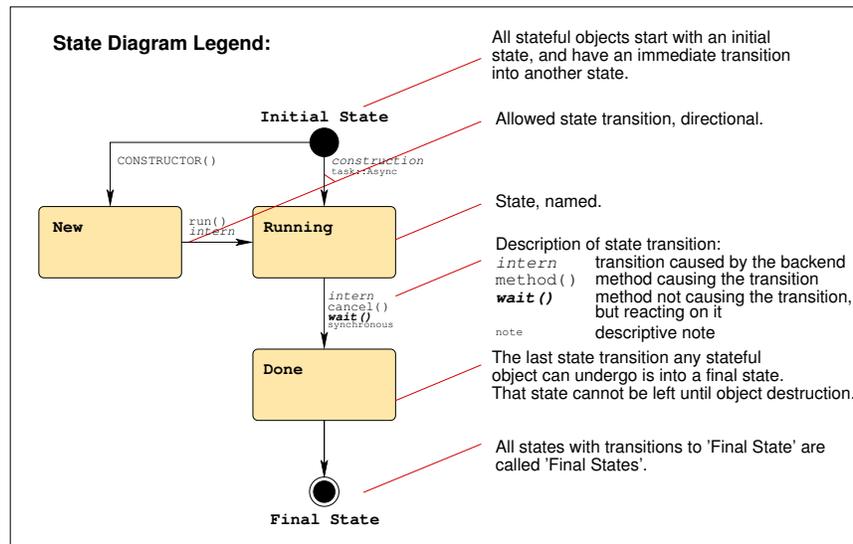


Figure 1: The SAGA state diagrams follow the notations shown here.

plications, 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** 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 implementa-

tion SHOULD always allow for application level consistency enforcement, for example by use of application level locks and mutexes.

2.9 Optimizing Implementations, Latency Hiding

Distributed applications are usually very sensitive 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 3.7) 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. 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 3.10.

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:

- As different SAGA implementations bind to different middleware, that middleware might need configuration information, such as the location of a GridRPC config file (see [14]), or the location of a service endpoint.
- If such configuration information are to be provided by the end user, the end user might face, eventually, a plethora of SAGA implementation specific configuration files, or environment variables, or other configuration mechanisms, which 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 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 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 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: server root:  /var/ftp/pub/  
HTTP Server: 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 users 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 URL's between different protocols (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 URL's succeeds, there is no guarantee that the referred file is actually available via this protocol, with the same permissions – 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. 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.
3. 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.
4. 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 can't be expected to work.
5. Any other error related to the URL (e.g. file at service is not available) MUST be indicated by the exceptions as listed in the method specifications in this document.

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.

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.

3 SAGA API Specification

The SAGA API consists of a number of interface and class specifications. The relation between these is shown in Figure 2 on Page 30. This figure also marks which interfaces are dominating the SAGA look-and-feel, and which classes are combined to packages.

The remainder of this section forms the main normative part of the SAGA API specification. It has one subsection for each package, starting with those interfaces that define the SAGA look-and-feel (top level interfaces first), followed by the various capability providing packages: job management, name space management, file management, replica management, stream, and remote procedure call.

SAGA Look & Feel API Packages

The SAGA Look & Feel is defined by a number of classes and interfaces which ensure the non-functional properties of the SAGA API (see [13] 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, however, not guaranteed to be the case. In particular Monitoring and Steering, but also asynchronous operations, may need explicit support from the backend system. As such, methods in these three packages **MUST** be expected to throw a **NotImplemented** exception, in accordance with the SAGA implementation compliance guidelines given in the introduction. 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.

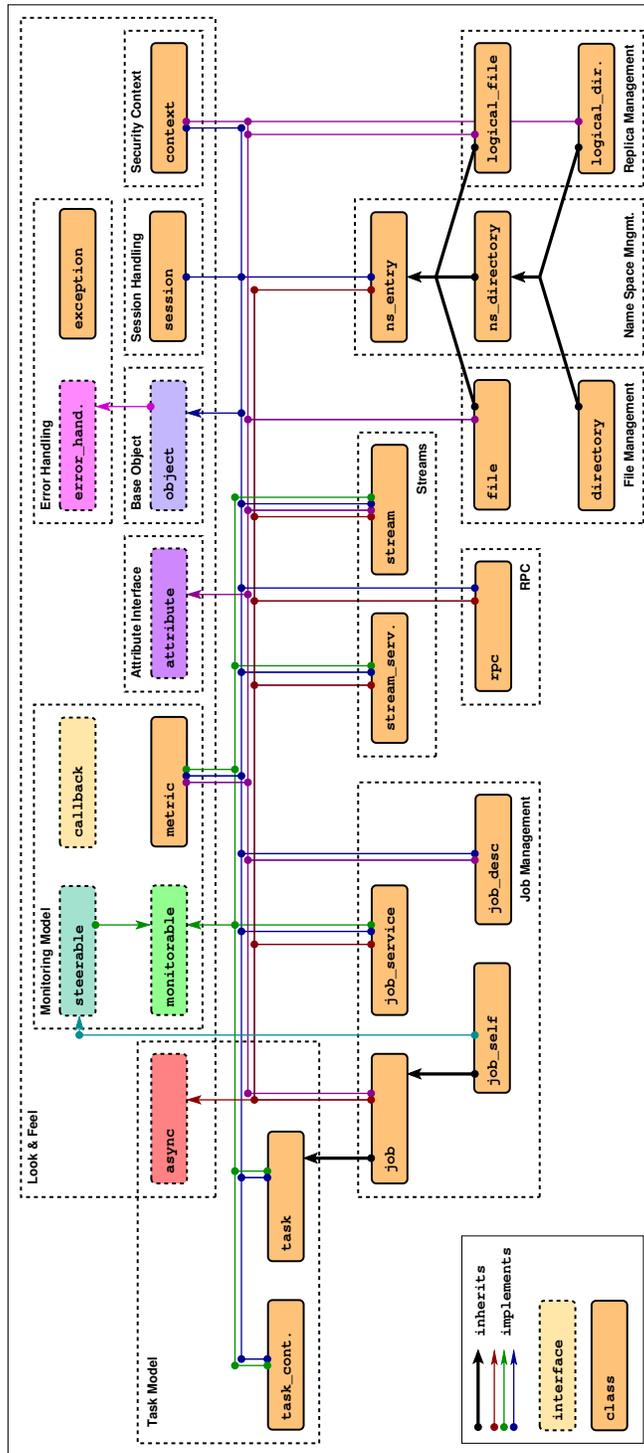


Figure 2: The SAGA class and interface hierarchy

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`, which allows a user of the API to query for the latest error associated with a saga object. In languages with exception facilities, such as Java, C++ and Perl, the language binding may allow exceptions to be thrown *instead*. 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.

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.

Some API methods return POSIX `errno` codes for errors. This is true 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, the user `errno` codes are to be used as they are used in POSIX.1.

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      object,
                     in string      message);
    DESTRUCTOR       (void);

    what              (out string      message);
    get_message       (out string      message);
    get_object        (out Object      object);
```

```
    }

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

3.1.2 Details

SAGA provides a set of well defined error states (exceptions) which MUST be supported by the implementation. As to whether 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 if the application really needs information from that file).

Several SAGA methods do not raise exceptions on certain error conditions, but return an error code. For example `file.read()`, might return an error code indicating that a non-blocking I/O does not have any data 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.

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 specification defines the set of allowed exceptions for each method explicitly – that set is normative.

The SAGA specification defines which exceptions can be thrown by which method. Depending on the implementation however, other exceptions can be thrown as well. For example, a certain implementation might have authorization as an attribute setting, and could throw an `AuthorizationFailed` exception on attempts to write that attribute – even though it is not specified in the SAGA specification. New SAGA exception types however, SHOULD NOT be defined by the implementation.

Listed exceptions are either derived from the base SAGA exception types or are error codes with that specific name etc. These exact rendering is language binding specific; for details, see the language bindings.

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.

The exception types defined in SAGA are listed below. This list 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.

NotImplemented:

If a method is specified in the SAGA API, but cannot be provided by a specific SAGA implementation, this exception MUST be thrown. See also the notes about compliant implementations in the instruction.

Example:

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

IncorrectURL:

This exception is thrown if a method is invoked with an URL argument that could not be handled. This error specifically indicates that an implementation can not handle the specified protocol, or 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 notes to 'The URL Problem' in the introduction.

Example:

- An implementation based on `gridftp` might be unable to handle http based URLs sensibly, and might be unable to translate them into ftp based URLs internally. The implementation should then throw an `IncorrectURL` exception if it encounters an http based URL.

IncorrectSession:

A method was invoked which effects two object instances which belong to different SAGA sessions.

Example:

- a stream is created in one session, and passed to the asynchronous version of the `serve()` method of the `stream_server` in another session.

AuthenticationFailed:

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

Example:

- 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.

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.

Example:

- 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.

PermissionDenied:

A 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.

Example:

- 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.

Notes:

- 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 on backend related authorization failures (which can often be resolved on user level).
- The AuthorizationFailed exception SHOULD be thrown when the 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.

BadParameter:

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

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 write on a file which is opened read only
- calling run on a task which was canceled
- calling resume on a job which is not suspended

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 attribute asked for is not supported
- a job asked for is not known by the backend

ReadOnly:

A attribute or metric was attempted to be changed but is read-only, e.g. is provided only for informational purposes. That exception does NOT apply for files or streams which are in incorrect state (i.e. not readable or writable) - that would cause an IncorrectState exception.

Examples:

- attempt to change or set a ReadOnly attribute
- attempt to change or update a ReadOnly metric

Timeout:

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

Examples:

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

Examples:

- a once open file is not available right now
- a backend response cannot be parsed
- a remote procedure call failed due to invalid input parameter
- 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 object,
out exception e);

Inputs: object: the object associated with the
exception.

Outputs: e: the newly created exception

Throws: -

- DESTRUCTOR

Purpose: destroy the exception

Format: DESTRUCTOR (in exception e);

Inputs: e the exception to destroy

Outputs: -

Throws: -

- what

what is an alias for `get_message`.

- get_message

Purpose: gets the message associated with an exception

Format: `get_message` (out string message);
Inputs: -
Outputs: `message` the error message
Throws: -
Notes: - the returned string MUST be formatted as described above.

- `get_object`

Purpose: gets the SAGA object associated with exception
Format: `get_object` (out object o);
Inputs: -
Outputs: `o`: the object associated with the exception
Throws: `NoSuccess`
Throws: -
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, or on `task.rethrow()`, the object is the one which the task was created from.
- an 'IncorrectState' 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.

3.1.3 Examples

Code Example

```
1 // c++ example
2 int main ()
3 {
4     try
5     {
6         saga::file f ("file://localhost/etc/passwd");
7         f.copy ("file:///usr/tmp/passwd.bak");
8     }
9
10    catch ( const saga::exception::PermissionDenied & e )
11    {
12        std::cerr << "SAGA error: No Permissions!" << std::endl;
13    }
14
15    catch ( const saga::exception & e )
```

```
16     {  
17         std::cerr << "SAGA error: " << e.what () << std::endl;  
18     }  
19  
20     return (0);  
21 }
```

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 [11].

3.2.1 Specification

```
package saga.object
{
  enum object_type
  {
    Unknown          = -1,
    Exception        =  1,
    Session          =  2,
    Context          =  3,
    Task             =  4,
    TaskContainer    =  5,
    Metric           =  6,
    NSEntry          =  7,
    NSDirectory      =  8,
    File             =  9,
    Directory        = 10,
    LogicalFile      = 11,
    LogicalDirectory = 12,
    JobDescription   = 13,
    JobServer        = 14,
    Job              = 15,
    StreamServer     = 16,
    Stream           = 17,
    Multiplexer      = 18
  }

  interface object : implements-all saga::error-handler
  {
    get_id      (out string      id      );
    get_type    (out object_type type  );
  }
}
```

```

        get_session (out session    session);

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

```

3.2.2 Details

```

class object:
-----

- get_id:
  Purpose: query the object ID
  Format:  get_id          (out string id);
  Inputs:  -
  Outputs: id              uuid for the object
  Throws:  -

- get_type:
  Purpose: query the object type
  Format:  get_type        (out object_type type);
  Inputs:  -
  Outputs: type           type of object
  Throws:  -

- get_session:
  Purpose: query the objects session
  Format:  get_session      (out session s);
  Inputs:  -
  Outputs: s                session of object
  Throws:  NoSuccess
  Notes   - if no specific session was attached to the
            object on creation time, the default SAGA
            session is returned.
            - some objects don't have sessions attached,
              such as job_description, task, metric, and the
              session object itself. For such objects, the
              method raises a 'NoSuccess' exception.

        // deep copy:

```

```
-----  
  
- clone:  
  Purpose: deep copy the object  
  Format: clone (out object clone);  
  Inputs: -  
  Outputs: clone the deep copied object  
  Throws: NotImplemented  
          NoSuccess  
  Throws: -  
  Notes - that method is overloaded by all classes  
         which implement saga::object, and returns  
         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 Introduction
```

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 buf1[100];  
10 char buf2[100];  
11 char buf[100];  
12  
13 // create map  
14 std::map <saga::task, saga::object> tmap;  
15  
16 // create objects, and map  
17 saga::file f (url[1]);  
18 saga::stream s (url[2]);  
19  
20 s.connect ();  
21  
22 // create tasks for reading first 100 bytes ...  
23 saga::task t1 = f.read <saga::task> (100, buf1, &out);
```

```
24  saga::task t2 = s.read <saga::task> (100, buf2, &out);
25
26  // ... and store in map
27  tmap[t1] = f;
28  tmap[t2] = s;
29
30  // create and fill the task container ...
31  saga::task_container tc;
32
33  tc.add (t1);
34  tc.add (t2);
35
36  // ... and wait who gets done first
37  while ( saga::task t = tc.wait () )
38  {
39      // depending on type, skip 100 byte then create a
40      // new task for the next read, and re-add to the tc
41
42      // store result
43
44
45      switch ( tmap[t].get_type () )
46      {
47          case saga::object::File :
48              // store result
49              buf = buf1;
50
51              // skip for file type (sync seek)
52              saga::file (tmap[t]).seek (100, SEEK_SET);
53
54              // create a new read task
55              tc.add (saga::file (tmap[t]).read <saga::task>
56                      (100, buf1, &out))
57
58              break;
59
60          case saga::object::Stream :
61              // store result
62              buf = buf2;
63
64
65              // skip for stream type (sync read and ignore)
66              saga::stream (tmap[t]).read (100, NULL);
67
68              // create a new read task
69              tc.add (saga::stream (tmap[t]).read <saga::task>
70                      (100, buf2, &out))
71
72              break;
73
```

```
74     default:
75         throw saga::exception ("Something is terribly wrong!");
76     }
77
78     std::cout << "found: '" << out << "'\n";
79
80
81     // tc is filled again, we run forever, read/seeking from
82     // whoever we find after the wait.
83 }
```

3.3 SAGA Session Handling

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 3.4).

3.3.1 Specification

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

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

3.3.2 Details

```
class session:
-----
```

Almost all saga objects are created in a SAGA session, and are associated with that (and only that) 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 handle is omitted as first parameter, a default session handle is used, with default security context(s) attached.

Example (c++):

```
// create a file object in a specific session:
saga::file f (session, url);

// create a file object in the default session:
saga::file f (url);
```

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

- `saga::context`
- `saga::job_description`
- `saga::metric`
- `saga::exception`
- `saga::tasks`
- `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, as that object can be retrieved from a `saga::task` instance, the `saga::session` instance is indirectly available.

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 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.

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 a `IncorrectSession` exception if 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 used backends.

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 removed.

Independent of any explicitly attached `saga::context` instances, a call to `list_contexts()` MUST include the default `saga::context` instances in the returned list.

Default `saga::context` instances on a session can be removed from a session, with a call to `remove_context()`.

A SAGA implementation MUST document what default context instances it may create and attach to a `saga::session`. That set MAY change during runtime, but must not be changed once a `saga::session` instance was created. E.g., two `saga::session` instances might have different default `saga::context` instances attached. Both sessions however will have these attached for their complete lifetime.

- CONSTRUCTOR

Purpose: create the object

Format: CONSTRUCTOR (out session obj)

Inputs: -

Outputs: obj: the newly created object

Throws: -

Notes: - The created session has the default context instances attached.

- DESTRUCTOR

Purpose: destroy the object
Format: DESTRUCTOR (in session obj)
Inputs: obj: the object to destroy
Outputs: -
Throws: -

- add_context
Purpose: attach a security context to a session handle
Format: add_context (in context context);
Inputs: context Security context to add
Outputs: -
Throws: -
PostCond: - the added context is deep copied
- if the session has already a context attached which has the exactly same set of attribute values as the parameter context, no action is taken.

- remove_context
Purpose: detach a security context from a session handle
Format: remove_context (in context context);
Inputs: -
Outputs: context Security context to remove
Throws: DoesNotExist
PostCond: -
Notes: - this methods removes all contexts on the session which have the exactly 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: -
Outputs: contexts list of contexts of this session
Throws: -
Note: - 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.
-

3.3.3 Examples

Code Example

```

1  // c++ example
2  saga::session s;
3  saga::context c (saga::context::Globus);
4
5  s.add_context (c);
6
7  saga::directory d (s, "gsiftp://remote.net/tmp/");
8  saga::file      f = dir.open ("data.txt");
9
10 // file has same session attached as dir,
11 // and can use the same contexts
12 -----+
13 // c++ example
14 saga::task  t;
15 saga::session s;
16
17 {
18     saga::context c (saga::context::Globus);
19
20     s.add_context (c);
21
22     saga::file f (s, url);
23
24     t = f.copy <saga::task::Task> (target);
25
26     s.remove_context (c);
27 }
28 // As it leaves the scope, the gsi context gets 'destroyed'.
29 // However, the copy task and the file object however MAY
30 // continue to use the Globus context, as its destruction is
31 // actually delayed until the last object using it gets
32 // destroyed.
33
34 t.run (); // can still use the Globus context

```

3.4 SAGA Context

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 3.3 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 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 a 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 attribute interface. Exactly what attributes a context offers depends on its type. A context MUST issue an error if attributes not corresponding to its type are accessed.

For application level AAA (e.g. for streams, monitoring, steering), read only contexts are used to inform the application about the requestor identity. To support that, a number of specific attributes are available, as specified below. They are named "`<context_type>_Remote<attribute>`".

The lifetime of `saga::context` instances are defined by the lifetime of those `saga::session` instances that context is associated with, and of those SAGA objects which have been created in these sessions. For detailed information about lifetime management, see the introduction (sec. 2.5.3), and the description of the SAGA session class in section 3.3.

3.4.1 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::attribute
                // from object saga::error_handler
{
    CONSTRUCTOR (in context_type type,
                 out context      context);
    DESTRUCTOR  (in context      context);

    get_ctype   (out context_type type);
}
}

```

3.4.2 Details

```
class context:
-----
```

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

Unknown:

No attributes supported

Globus:

ReadWrite:

```

UserProxy      (/tmp/x509up_u<uid>)
CertDir        (/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>)
 CertDir (/etc/grid-security/certificates/)
 UserCert (\$HOME/.globus/usercert.pem)
 UserKey (\$HOME/.globus/userkey.pem)
 UserPass
ReadOnly:
 RemoteID
 RemoteHost
 RemotePort

SAML:

ReadWrite:
 UserName
 UserPass
ReadOnly:
 RemoteID
 RemoteHost
 RemotePort

Unicore:

ReadWrite:
 UserCert (\$HOME/.unicore/keystore)
 UserPass
ReadOnly:
 RemoteID
 RemoteHost
 RemotePort

SSH:

ReadWrite:
 CertDir (\$HOME/.ssh/)
 UserCert (\$HOME/.ssh/id_dsa.pub)
 UserPass (\$HOME/.ssh/id_dsa)
ReadOnly:
 RemoteID

RemoteHost
RemotePort

Kerberos:

ReadWrite:
Server (localhost)
UserName
UserPass
ReadOnly:
RemoteID
RemoteHost
RemotePort

UserPass:

ReadWrite:
UserName
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 context);
Inputs: type type of context
Outputs: context the newly created context
Throws: BadParameter
Notes: - BadParameter is thrown if a context type is
not supported (NOT NotImplemented).

- DESTRUCTOR:

Purpose: destroy a security context
Format: DESTRUCTOR (in context context);
Inputs: context the context to destroy
Outputs: -
Throws: -

- `get_ctype`:
Purpose: query the context type
Format: `get_ctype` (out context_type type);
Inputs: -
Outputs: type type of context
Throws: -

3.4.3 Examples

Code Example

```
1 // c++ example
2 // see notes to the URL problem in the introduction!
3
4 saga::context c_1 (saga::context::SSH); // default attribs
5 saga::context c_2 (saga::context::FTP);
6
7 c_2.set_attribute ("ID", "myself");
8 c_2.set_attribute ("Pass", "secret");
9
10 saga::session s;
11 s.add_context (c_1);
12 s.add_context (c_2);
13
14 saga::file f ("any://remote.net/tmp/data.txt", s);
15
16 // file can be accessed now via ssh or ftp
17 f.copy ("data.bak");
```

3.5 SAGA Attribute Interface

There are various places in the SAGA API where attributes need to be associated with objects, for instance for job descriptions and metrics. The 'Attribute' 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 naming 'Attribute' is somewhat misleading: it seems to imply that an object implementing this interface **IS-A** attribute. What we actually mean is that an object implementing this interface **HAS** attributes. In the want of a better name, we left it 'Attribute', but implementers and users should be aware of the actual meaning (The proper interface naming would be 'attributable', which sounds awkward).

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

3.5.1 Specification

```
package saga.attribute
{
    interface attribute
    {
        // 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 string      kpat,
```

```

        in string      vpat,
        out array<string> keys);
    attribute_equals  (in string      key,
                     in string      val,
                     out bool       test);
    attribute_exists  (in string      key,
                     out bool       test);
    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);
}
}

```

3.5.2 Details

The attribute interface in SAGA provides a uniform paradigm to set and query parameters and properties of SAGA objects. Although the attribute 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 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

```
saga::stream->set_attribute ("BufferSize", "1024");
```

a language binding might allow

```
saga::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 of attributes cannot be handled transparently, e.g. by overloading. Bindings for languages such as Python, Perl and C++ CAN hide that 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 MUST be preserved by the SAGA implementation. The equals method does also rely on ordering (i.e. "one" "two" does not equal "two" "one").

Attributes are expressed as string values, however, they do have 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, attribute 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 `"%Lf"`, 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** point type attributes are expressed as they would in result of a `printf` of the format `"%lld"`, 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 format the string as a **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.

3.5.3 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 such a definition is (from the Metric object):

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

    // Attributes:
    // name: Name
    // desc: name of metric
    // mode: ReadOnly
    // type: Scalar 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 given as '-', the attribute is not set by default. 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 a attribute which got removed ('dead attribute') **MUST** throw an `IncorrectState` exception. However, dead attributes **MUST** stay available for read access. The SAGA implementation **MUST NOT** change that attributes 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 for attribute keys and values, and returns a list of keys for those attributes which match any one of the specified pattern. The allowed patterns are the same as defined as wildcards in the description of the SAGA name space objects, and are to be formatted as: `<key-pattern>=<value-pattern>`.

```
interface attribute:
```

```
-----
```

```
- 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
```

Outputs: -

Throws: AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
ReadOnly
DoesNotExist
Timeout
NoSuccess

Notes: - a empty string means to set an empty value (the attribute is not removed).
- the attribute is created, if it does not exist
- a 'ReadOnly' 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, and 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.
- 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 attributes value

Format: `get_attribute` (in string key,
out string value);

Inputs: key: attribute key

Outputs: value: value of the attribute

Throws: AuthenticationFailed
AuthorizationFailed
PermissionDenied

DoesNotExist

Timeout

NoSuccess

- Notes:
- queying 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 resepective 'AuthenticationFailed', 'AuthorizationFailed', and 'PermissionDenied' exceptions are thrown. For example, and 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.
 - 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 values for the
attribute

Outputs: -

Throws: AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
ReadOnly
DoesNotExist
Timeout
NoSuccess

Notes: - the notes to the `set_attribute()` method apply.

- `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

Outputs: values: array of values of the attribute.

Throws: AuthenticationFailed
AuthorizationFailed
PermissionDenied
DoesNotExist
Timeout
NoSuccess

Notes: - the notes to the `get_attribute()` method apply.

- `remove_attribute`

Purpose: removes an attribute.
Format: `remove_attribute` (in string key);
Inputs: key: attribute to be removed
Outputs: -
Throws: AuthenticationFailed
AuthorizationFailed
PermissionDenied
DoesNotExist
ReadOnly
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 'ReadOnly' 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: -
Outputs: keys: existing attribute keys
Throws: AuthenticationFailed
AuthorizationFailed
PermissionDenied
Timeout
NoSuccess

Notes: - exceptions have the same semantics as defined

for the `get_attribute()` method description.

- `find_attributes`
 - Purpose: find matching attributes.
 - Format: `find_attributes` (in `array<string>` `pattern`,
out `array<string>` `keys`);
 - Inputs: `pattern`: key/value pattern
 - Outputs: `keys`: matching attribute keys
 - Throws: `BadParameter`
`AuthenticationFailed`
`AuthorizationFailed`
`PermissionDenied`
`Timeout`
`NoSuccess`
 - Note:
 - 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_equals`
 - Purpose:
 - Format: `attribute_equals` (in `string` `key`,
in `string` `val`,
out `bool` `test`);
 - Inputs: `key`: attribute key
`val`: val to compare against
 - Outputs: `test`: `bool` indicating success
 - Throws: `DoesNotExist`
`AuthenticationFailed`
`AuthorizationFailed`
`PermissionDenied`
`Timeout`
`NoSuccess`
 - Notes:
 - this method returns `TRUE` if the attribute identified by `key` has the value identified by `val`.
 - for vector attributes, the value has to be specified as comma delimited concatenated string of the vector elements.
 - exceptions have the same semantics as defined for the `get_attribute()` method description.

- `attribute_exists`
 - Purpose:
 - Format: `attribute_exists (in string key, out bool test);`
 - Inputs: `key: attribute key`
 - Outputs: `test bool indicating success`
 - Throws: `AuthenticationFailed`
`AuthorizationFailed`
`PermissionDenied`
`DoesNotExist`
`Timeout`
`NoSuccess`
 - Notes:
 - This method returns TRUE if the attribute identified by key exists.
 - This method returns FALSE if the attribute identified by key does not exist, and does NOT throw a `DoesNotExist` exception.
 - exceptions have the same semantics as defined for the `get_attribute()` method description.

- `attribute_is_readonly`
 - Purpose:
 - Format: `attribute_is_readonly(in string key, out bool test);`
 - Inputs: `key: attribute key`
 - Outputs: `test bool indicating success`
 - Throws: `AuthenticationFailed`
`AuthorizationFailed`
`PermissionDenied`
`DoesNotExist`
`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:
 - Format: `attribute_is_writable(in string key, out bool test);`

Inputs: key: attribute key
Outputs: test bool indicating success
Throws: AuthenticationFailed
AuthorizationFailed
PermissionDenied
DoesNotExist
Timeout
NoSuccess
Notes: - This method returns TRUE if the attribute identified by the key exists, and can be changed by `set_attribute()` and `set_vector_attribute()`.
- exceptions have the same semantics as defined for the `get_attribute()` method description.

- `attribute_is_removable`
Purpose:
Format: `attribute_is_removable` (in string key, out bool test);
Inputs: key: attribute key
Outputs: test bool indicating success
Throws: AuthenticationFailed
AuthorizationFailed
PermissionDenied
DoesNotExist
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:
Format: `attribute_is_vector` (in string key, out bool test);
Inputs: key: attribute key
Outputs: test bool indicating if attribute is scalar (false) or vector (true)
Throws: AuthenticationFailed
AuthorizationFailed
PermissionDenied

DoesNotExist

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.
-

3.5.4 Examples

Code Example

```
1 // c++ example:
2 job_definition d;
3
4 // vector attributes
5 d.set_attribute ("ExecutionHosts", "host_1, host_2");
6
7 // scalar attribute
8 d.set_attribute ("MemoryUsage", "1024 MB");
9
10 ...
```

3.6 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, which represent values to be monitored.

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 by the ability to push values back to the monitored entity, i.e. to introduce writable metrics (see `fire()`).

3.6.1 Specification

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

  // a metric represents an entity / value to be monitored.
  class metric : implements saga::object
                implements saga::attribute
                // 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      metric);
DESTRUCTOR      (in metric      metric);

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

```
    // value: see description of value formatting 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);
}
}
```

3.6.2 Details

interface callback:

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

on 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 necessary 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 a 'NotAuthorized' 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 exception.

For example, if a stream_server instance invokes a callback on a ClientConnect metric, and the cb method raises a 'NotAuthorized' 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 a 'NotAuthorized' 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, 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 = Unknown,
                             out bool       keep);
  Inputs: mt:               the saga monitorable object
                             which cause the callback
                             invocation
           metric:          the metric causing the
```

callback invocation

ctx: the context associated with the callback causing entity

Outputs: keep: indicates if callback stays registered

Throws: AuthorizationFailed

Notes:

- 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.
- '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 object the metric 'metric' belonged 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.
- the passed context MUST be authenticated.
- 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 (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 multiple times.
- a callback added to exactly one metric exactly once is guaranteed to be active at most once at any given time.

```
class metric:
```

```
-----
```

The fundamental object introduced in this package is a metric. A metric represents an observable, 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:      "Read", "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
- job.temperature    // a custom observable on a job
```

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 "Read" to "Final".

The name attribute MUST be interpreted case insensitive: An implementation MAY change that attribute to 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.

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 such a definition is (from the SAGA stream object):

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

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

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 error message **MUST** state "Metric <name> 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 Life Time:

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. However, existing class instances of a dead metric MUST stay valid, and expose the same life time as any other 'life 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. Callback 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, callback 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 are available, a context of type 'Unknown' is passed, which has no attributes attached.

A callback can evaluate the passed context, and throw a 'NotAuthorized' 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 metric
desc: description of metric
mode: mode of metric

```

        unit:          unit of metric value
        type:          type of metric
        value:         initial value of metric
Outputs: obj:         the newly created object
Throws: NotImplented
        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
        initialise 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
  Outputs: -
  Throws:  -
  Notes:  - on destruction, all callbacks get removed
        - if a callback is active at time of
        destruction, the destructor MAY block until
        that callback returns. No other callbacks
        get activated during that block.

// manage callbacks on the metric
- add_callback
  Purpose: add asynchron notifier callback to watch metric
        changes
  Format:  add_callback    (in callback cb,
                          out int      cookie);

```

Inputs: cb: callback class instance
Outputs: cookie: handle for this callback,
to be used for removal

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
IncorrectState
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 backend cannot guarantee that the
callback gets invoked 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
Outputs: -
Throws: NotImplemented
BadParameter
Timeout
NoSuccess
Notes: - 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, authorization or permission faults are expected
-
- fire
 - Purpose: push a new metric value to the backend
 - Format: fire (void);
 - Inputs: -
 - Outputs: -
 - Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
IncorrectState
ReadOnly
Timeout
NoSuccess
 - Notes:
 - 'IncorrectState' is thrown if the metric is Final.
 - 'ReadOnly' is thrown if the metric is not Writable -- That holds also for a once writable metric which was flagged Final. To catch race condition triggered exceptions, each fire should be try'ed/catched.
 - 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 are inviced.
 - 'AuthenticationFailed', 'AuthorizationFailed'

- or 'PermissionDenied' may get thrown if the current 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 on 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:  -
  Outputs: names:          array of names identifying
                          the metrics associated with
                          the object instance

  Throws:  NotImplemented
          AuthenticationFailed
          AuthorizationFailed
          PermissionDenied
          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
          be actually 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 metric to be returned
  Outputs: metric:        metric instance identified by
                          name
  Throws:  NotImplemented
          AuthenticationFailed
          AuthorizationFailed
          PermissionDenied
          DoesNotExist
          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 metric to which cb
```

```

                                is to be added
                                reference of callback class
                                instance to be registered
                                handle to be used for removal
                                of the callback
Outputs:  cb:
Outputs:  cookie:
Throws:   NotImplemented
          AuthenticationFailed
          AuthorizationFailed
          PermissionDenied
          DoesNotExist
          Timeout
          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 metric for which
                              cb is to be removed
          cookie:            identifies the cb to be
                              removed
Throws:   NotImplemented
          DoesNotExist
          BadParameter
          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
  Outputs: success:        indicates success
  Throws:  NotImplemented
           AuthenticationFailed
           AuthorizationFailed
           PermissionDenied
           AlreadyExists
           ReadOnly
           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 (state of metric is not
      changed)
    - a 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, a 'ReadOnly'
      exception is thrown.

- remove_metric
  Purpose: remove a metric instance
  Format:  remove_metric   (in string name);
```

Inputs: name: identifies metric to be removed

Outputs: -

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
DoesNotExist
ReadOnly
Timeout
NoSuccess

Notes: - only previously added metrics can be removed; default (saga defined or implementation specific) metrics 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, a 'ReadOnly' 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 metric to be fired

Outputs: -

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
IncorrectState
DoesNotExist
ReadOnly
Timeout

- 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 an `'ReadOnly'` metric results in a `'ReadOnly'` exception.
 - an attempt to fire a `'Final'` metric results in a `'IncorrectState'` exception.
-

3.6.3 Examples

Code Example

```
1  callback example: trace all task state changes:
2  -----
3
4  // c++ example
5  // callback definition
6  class trace_cb : public saga::callback
7  {
8  public:
9      bool cb (saga::monitorable mt,
10             saga::metric      m,
11             saga::context      c)
12      {
13          std::cout << "metric " << m.get_attribute ("name")
14                  << " fired." << std::endl;
15          return true; // stay registered
16      }
17  }
18
19 // the application
20 int main ()
21 {
```

```
22     ...
23
24     // if the callback defined above is added to all known
25     // metrics of all saga objects, a continous trace of state
26     // changes of these saga objects will be written to stdout
27     trace_cb cb;
28
29     saga::job j = ...
30
31     j.add_callback ("state", cb);
32
33     ...
34 }
35
36
37 monitoring example: monitor a write task
38 -----
39
40 // c++ example for task state monitoring
41 class write_metric_cb : public saga::callback
42 {
43     private:
44         saga::task t_;
45
46     public:
47         write_metric_cb (const saga::task & t) { t_ = t; }
48
49         bool cb (saga::monitorable mt,
50                 saga::metric      m,
51                 saga::context      c)
52         {
53             std::cout << "bytes written: "
54                       << m.get_attribute ("value")
55                       << std::endl;
56
57             std::cout << "task state:  "
58                       << t_.t_state ()
59                       << std::endl;
60
61             return (false); // keep calback registered
62         }
63 };
64
65 int main (int argc, char** argv)
66 {
67     ssize_t      len = 0;
68     std::string str ("Hello SAGA\n");
69     std::string url (argv[1]);
70
71     saga::file   f (url);
```

```
72     saga::task t = f.write <saga::task> (str, &len);
73
74     // assume that file has a 'progress' metric indicating
75     // the number of bytes already written. In general,
76     // the list of metric names has to be searched for an
77     // interesting metric, unless it is a default metric as
78     // specified in the SAGA spec.
79
80     // create and add the callback instance
81     write_metric_callback cb (t);
82     f.add_callback ("progress", cb);
83
84     // wait until task is done, and give cb chance to get
85     // called a couple of times
86     t.wait ();
87 }
88
89
90 steering example: steer a remote job
91 -----
92
93 // c++ example
94 class observer_cb : public saga::metric::callback
95 {
96     private:
97         saga::task t;
98
99     public:
100         bool cb (saga::monitorable mt,
101                 saga::metric m,
102                 saga::context c)
103         {
104             int val = atoi ( m.get_attribute ("value") );
105
106             std::cout << "the new value is"
107                       << atoi ( m.get_attribute ("value") )
108                       << std::endl;
109
110             return (false); // keep callback registered
111         }
112 };
113
114 // the steering application
115 int main (int argc, char** argv)
116 {
117     saga::job_service js;
118
119     saga::job j = js.run ("remote.host.net",
120                          "my_remote_application");
121 }
```

```
122 // Assume that job has a 'param_1' metric representing
123 // a integer parameter for the remote application.
124 // In general, one has to list the metrics available on
125 // job, with list_metric, and search for an interesting
126 // metric. However, we assume here that we know that
127 // metric exists. So we just add an observer callback
128 // to the 'param_1' metric - that causes the
129 // asynchronous printout of any changes to the value
130 // of that metric
131
132 observer_cb cb;
133 j.add_callback ("param_1", cb);
134
135 // then we get metric for active steering
136 saga::metric m = j.get_metric ("param_1");
137
138 for ( int i = 0; i < 10; i++ )
139 {
140 // if param_1 is ReadOnly, set_value would throw
141 // 'ReadOnly' - it would net be usable for
142 // steering then.
143 m.set_attribute ("value", std::string (i));
144
145 // push the pending change out to the receiver
146 m.fire ();
147
148 // callback should get called NOW + 2*latency
149 // That means fire REQUESTS the value change, but only
150 // the remote job can CHANGE the value - that change
151 // needs then reporting back to us.
152
153 // give steered application some time to react
154 sleep (1);
155 }
156 }
157
158
159
160 steering example: BE a steerable job
161 -----
162
163 // c++ example
164 //
165 // the example shows a job which
166 // - creates a metric to expose a Float steerable
167 // parameter
168 // - on each change of that parameter computes a
169 // new isosurface
170 //
171 // callback - on any change of the metric value, e.g. due to
```

```
172 // steering from a remote GUI application, a new iso surface
173 // is computed
174 class my_cb : public saga::callback
175 {
176     public:
177         // the callback gets called on any
178         bool cb (saga::monitorable mt,
179                 saga::metric      m,
180                 saga::context      c)
181         {
182             // get the new iso-value
183             float iso = atof (m.get_attribute ("value"));
184
185             // compute an isosurface with that iso-value
186             compute_iso (iso);
187
188             // keep this callback alive, and get called again on
189             // the next metric event.
190             return (false);
191         }
192     }
193
194 int main ()
195 {
196     // create a metric for the iso-value of an isosurfacers
197     saga::metric m ("application.isosurfacers.isovalue",
198                   "iso-value of the isosurfacers",
199                   "ReadWrite", // steerable
200                   "",          // no unit
201                   "Float",    // data type
202                   "1.0");     // initial value
203
204     // add the callback which reacts on changes of the
205     // metric's value (returned cookie is ignored)
206     my_cb cb;
207     m.add_callback (cb);
208
209     // get job handle for myself
210     saga::self self;
211
212     // add metric to myself
213     self.add_metric (m);
214
215     /*
216     // the callback could also have been added with:
217     self.add_callback ("application.isosurfacers.isovalue", cb);
218     */
219
220     // now others can 'see' the metric, e.g. via
221     // job.list_metrics ();
```

```
222
223 // compute isosurfaces for the next 10 minutes -
224 // the real work is done in the callback, on incoming
225 // requests (i.e. steering events).
226 sleep (600);
227
228 // on object (self) destruction, metrics and callback
229 // objects are destroyed as well
230 return (0);
231 }
232
233
234
235 monitoring example: callback for stream connects
236 -----
237
238 // c++ example
239 //
240 // callback class which accepts an incoming client
241 // connection, and then un-registered itself. So, it
242 // accepts exactly one client, and needs to be re-registered
243 // to accept another client.
244 class my_cb : public saga::callback
245 {
246     privat:
247         // we keep a stream server and a single client stream
248         saga::stream_server ss_;
249         saga::stream s_;
250
251     public:
252         // constructor initialises these (note that the
253         // client stream should be not connected at this
254         // point)
255         my_cb (saga::stream_server ss,
256              saga::stream s )
257         {
258             ss_ = ss;
259             s_ = s;
260         }
261
262
263         // the callback gets called on any incoming client
264         // connection
265         bool cb (saga::monitorable mt,
266                saga::metric m,
267                saga::context c)
268         {
269             // the stream server got an event triggered, and
270             // should be able to create a client socket now.
271             s_ = ss_.wait ();
```

```
272         if ( s_.state == saga::stream::open )
273         {
274             // have a client stream, we are done
275             // don't call this cb again!
276             return (true);
277         }
278
279         // no valid client stream obtained: keep this
280         // callback alive, and get called again on the
281         // next event on ss_
282         return (false);
283     }
284 }
285 }
286
287 int main ()
288 {
289     // create a stream server, and an un-connected
290     // stream
291     saga::stream_server ss;
292     saga::stream      s;
293
294     // give both to our callback class, and register that
295     // callback with the 'client_connect' metric of the
296     // server. That causes the callback to be invoked on
297     // every change of that metric, i.e. on every event
298     // that changes that metric, i.e. on every client
299     // connect attempt.
300     my_cb cb (ss, s);
301     ss.add_callback ("client_connect", cb);
302
303     // now we serve incoming clients forever
304     while ( true )
305     {
306         // check if a new client is connected
307         // the stream state would then be Open
308         if ( s.state == saga::stream::Open )
309         {
310             // a client got conncted!
311             // handle open socket
312             s.write ("You say hello, I say good bye!\r\n", 32);
313
314             // and close stream
315             s.close ();
316
317             // the stream is not Open anymore. We re-add the
318             // callback, and hence wait for the next client
319             // to connect.
320             ss.add_callback ("client_connect", cb);
321         }
322     }
323 }
```

```
322     else
323     {
324         // no client yet, idle, or do something useful
325         sleep (1);
326     }
327 }
328
329 // we should never get here
330 return (-1);
331 }
```

3.7 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.

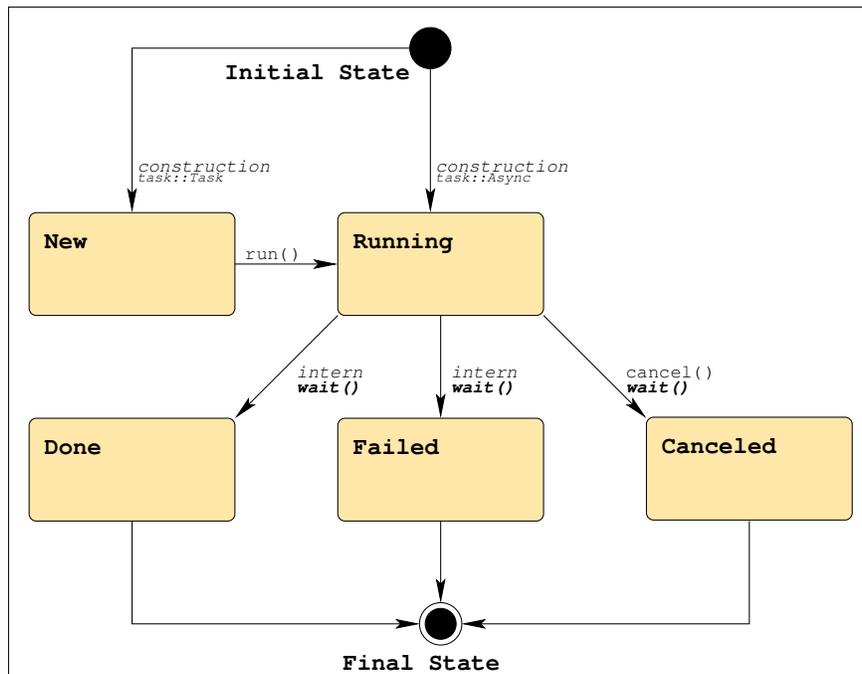


Figure 3: The SAGA task state model (See figure 1 for a description).

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 [8], which is in particular relevant to the (similar) job state model as described in section 3.8.

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 SAGA calls correspond to the normal method calls specified in the SAGA specification. The first `out` parameter specified (if any) is used as return value.
- The *asynchronous* version of SAGA calls has a different signature, and 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 cancelled.
- The *task* version of SAGA calls 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 synchronous calls, which have the same signature as asynchronous and task versions, but the returned task is in a final state (i.e. `run()` and `wait()` have been called on that task before returning).²
- `out` and `inout` parameters for asynchronous operations 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.
- `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.

Errors arising from synchronous method invocations on SAGA objects are, in general, flagged by exceptions, and can also 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 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 that 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.

²Note that state transitions for this type of method call are not shown in the state diagram – the diagram would essentially need to allow 'Done' as a initial 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. It is, however, not allowed to throw an `AuthorizationFailed`, `AuthenticationFailed` or `PermissionDenied` exception, as this spec assumes that the creator of the task can always inspect and control that task. Later versions of this API MAY change that, for example when they introduce persistent tasks which can survive the lifetime of a SAGA application.

3.7.1 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 to distinguish the three task-returning method calls.

Code Example

```
1 // c++ like example
2
3 // SAGA specification:
4 // read      (in  int          len_in,
5 //            inout array<byte> buffer,
6 //            out  int          len_out );
7
8 // synchronous version
9 ssize_t len_out = saga::file::read ( char * buffer,
10                                     size_t len_in );
11
12 // alternative synchronous version
13 saga::task t1 = saga::file::read <saga::task::Sync>
14                 ( char * buffer,
15                 size_t len_in,
16                 ssize_t & len_out);
17
18 // asynchronous version
19 saga::task t2 = saga::file::read <saga::task::ASync>
20                 ( char * buffer,
21                 size_t len_in,
```

```
22         ssize_t & len_out);
23
24 // asynchronous version
25 saga::task t3 = saga::file::read <saga::task::Task>
26         ( char    * buffer,
27         size_t    len_in,
28         ssize_t & len_out);
29
30 // t1 is in Done or Failed state
31 // t2 is in Running state
32 // t3 is in New state
```

A C language binding of this package might choose to use flags to distinguish 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 to not 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.

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.

3.7.2 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
    DESTRUCTOR (in task      obj);

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

    get_state  (out state     state);

    rethrow    (void);

    // Metric:
    // name: state
    // desc: "fires if on task state change, and
    //       has the literal value of the task
    //       state enum."
    // mode: Read
    // Unit: 1
    // Type: Int
    // Value: "0"
}
```

```

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

    add          (in task          task,
                 out int          cookie);
    remove      (in int          cookie);

    run          (void);
    cancel      (in float        timeout = 0.0);
    wait        (in float        timeout = -1.0,
                 int wait_mode   mode   = All,
                 out array<task> finished);

    list_tasks  (out array<int>   cookies);
    get_tasks   (out array<task> tasks);
    get_states  (out array<state> states);

    // Metric:
    //   name: State
    //   desc: fires on state changes of any task in
    //         container, and has the value of that
    //         tasks cookie.
    //   mode: Read
    //   unit: 1
    //   type: Enum
    //   value: "Unknown"
}
}

```

3.7.3 Details

```

enum state:
-----

```

A task can be in one of several possible states:

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.
This state 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.
This state corresponds to the BES state 'Canceled'.
This state is final.

Failed:

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

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 are also effectively created as jobs.

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

- CONSTRUCTOR
No constructor is available, as tasks get only created through asynchronous method calls.

- DESTRUCTOR
Purpose: destroy the object
Format: DESTRUCTOR (in task obj)
Inputs: obj: the object to destroy
Outputs: -
Throws: -
PostCond: - state is no longer shared with the creating object.
- a nonblocking cancel() was called on the task if it was in 'Running' or 'Suspended' state during destruction.

- run
Purpose: Start the asynchronous operation.
Format: run (void);
Inputs: -
Outputs: -
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
Outputs: done: indicating if the task is done running
Throws: NotImplemented
IncorrectState
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 unsuccessful (false) return value.
 - for timeout semantics, see Introduction
- cancel
- Purpose: Cancel the asynchronous operation.
- Format: cancel (in float timeout);
- Inputs: timeout: time for freeing resources
- Outputs: -
- PreCond: - task is in 'Running' state
- Throws: NotImplemented
IncorrectState
NoSuccess
- Notes:
- for resource deallocation semantics, see Introduction
 - 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 affect, 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 cancel for the task.
 - for timeout semantics, see Introduction

```
- get_state
Purpose: Get the state of the task.
Format:  get_state          (out state state);
Inputs:  -
Outputs: state:             state of the task.
Throws:  NotImplemented
         Timeout
         NoSuccess

Notes:   - a 'Timeout' or 'NoSuccess' exception indicates
         that the backend was not able to retrieve the
         task state.

- rethrow
Purpose: re-throw any exception a failed task caught.
Format:  throw (void);
Inputs:  -
Outputs: -
Throws:  NotImplemented
         IncorrectURL
         IncorrectSession
         AuthenticationFailed
         AuthorizationFailed
         PermissionDenied
         BadParameter
         IncorrectState
         AlreadyExists
         DoesNotExist
         ReadOnly
         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.
```

```
class task_container:
-----
```

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

When there are many asynchronous tasks it would be inefficient to invoke the `wait()` method on each one sequentially. The `task_container` class provides a mechanism to wait (amongst other operations) for a set of tasks.

- CONSTRUCTOR:

Purpose: create a task container

Format: CONSTRUCTOR (out task_container tc);

Inputs: -

Outputs: tc: newly created container

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

Outputs: -

Throws: -

Notes: - tasks in the task container during its destruction are not affected by its destruction, and, in particular, are not cancelled.

- add

Purpose: Add a task to a task_container.

Format: add (in task task);

Inputs: task: task to add to the task_container

Outputs: -

Throws: NotImplemented

Timeout

NoSuccess

Notes: - a task can be added more than once

- 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 task task);
 - Inputs: task: task to remove from the task_container
 - Outputs: -
 - Throws: NotImplemented
DoesNotExist
Timeout
NoSuccess
 - Notes:
 - if a task was added more than once, it must be removed the same number of times in order to leave no trace of it in the task container.
 - if the task 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.
-
- run
 - Purpose: Start all asynchronous operations in the container.
 - Format: run (void);
 - Inputs: -
 - Outputs: -
 - Throws: NotImplemented
IncorrectState
Timeout
NoSuccess
 - Notes:
 - run() will 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.
 - 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 float timeout,
in wait_mode mode = All,
out task done);
 - Inputs: timeout: seconds to wait
mode: wait for All or Any task
 - Outputs: done: finished task
 - Throws: NotImplemented
IncorrectState
NoSuccess
 - Notes:
 - if mode is 'All', the wait call returns only if all tasks in the container are finished, or on timeout, whatever 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 ( task = tc.wait (saga::task::Any) )
{
    ...
}
```
 - wait() will cause an 'IncorrectState' exception if any of the tasks in the container causes that exception on wait().
 - 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 Introduction

- cancel
 - Purpose: Cancel all the asynchronous operations in the container.
 - Format: cancel (in float timeout);
 - Inputs: timeout: time for freeing resources
 - Outputs: -

Throws: NotImplemented
IncorrectState
Timeout
NoSuccess

Throws: -

Notes: - see semantics of task cancel.
- if any task in the container is in the 'New' state, an 'IncorrectState' exception is thrown.
- a 'Timeout' or 'NoSuccess' exception indicates that the backend was not able to run 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.

- list_tasks

Purpose: List the tasks in the task task_container.

Format: list_tasks (out array<int> cookies);

Outputs: cookies: array of cookies for all tasks in task_container

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_tasks

Purpose: Get the tasks in the task task_container.

Format: get_tasks (out array<task> tasks);

Outputs: tasks: array of tasks in task_container

Throws: NotImplemented
ReadOnly
Timeout
NoSuccess

Throws: -

Notes: - the returned tasks are NOT removed from the task container.
- 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 `task_container`.
Format: `get_states` (out `array<state> states`);
Outputs: `states`: array of states for tasks in `task_container`
Throws: `NotImplemented`
`Timeout`
`NoSuccess`
Notes: - the returned list is not ordered
- a 'Timeout' or 'NoSuccess' exception indicates that the backend was not able to obtain the states of the tasks in the container.

3.7.4 Examples

Code Example

```
1 // c++ example, partly pseudocode
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 ( saga::exception e )
52         {
53             std::cout << "task failed: " << e.what () << std::endl;
54         }
55     }
56 }
```

SAGA Functional API Packages

The Functional SAGA API packages define the functional SAGA API scope, as motivated in the introduction and in [13].

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 authentication. 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. Additional, the conventions for the exceptions `NotImplemented` and `IncorrectURL` apply as described in section 3.

3.8 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 [5]. Much of this specification was taken directly from DRMAA specification [16], with many of the differences arising from an attempt to make the job API consistent with the overall SAGA API look&feel³.

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 [10] based keys, defines the job to be started, and its resource requirements. The job server represents a resource management endpoint which allows the starting and listing 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, and allows for a number of use cases which have the application actively interacting with the Grid infrastructure, for example to provide steering capabilities, to migrate itself, or to set job attributes.

The job class inherits the `saga::task` class 3.7, 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 jobs 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.

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

3.8.1 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.

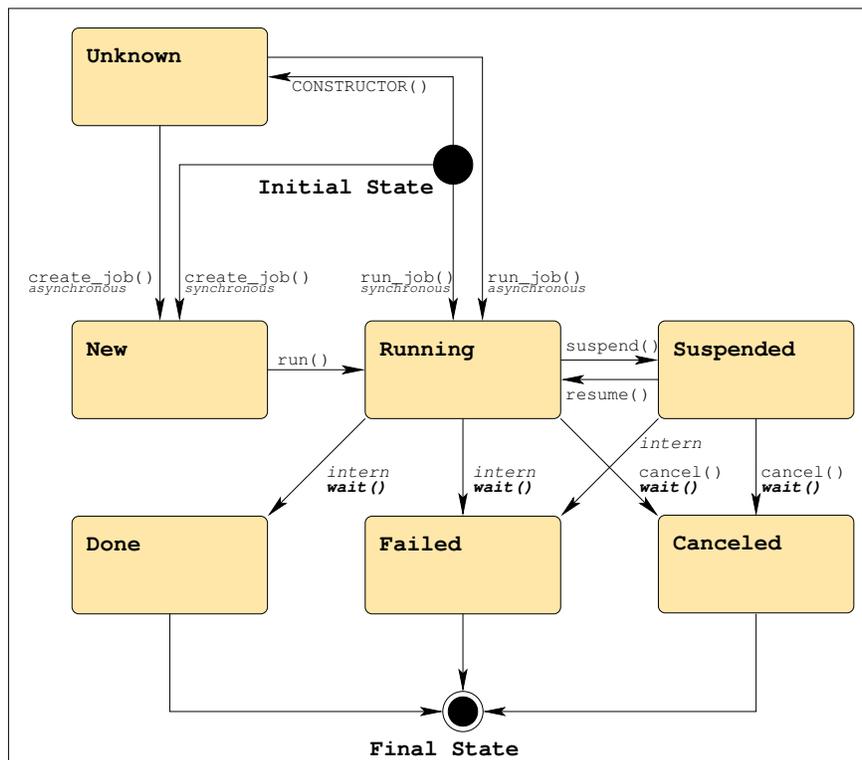


Figure 4: The SAGA job state model extends the SAGA task state model with 'Unknown' and 'Suspended' states, and additional transitions (See figure 1 for a legend).

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 [8], 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'). If no state details are available, the metric is still available, but it has always an empty string value.

3.8.2 Job Description Attributes

Although JSDL [2] based attribute names are used for job description, the API uses 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.

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 aren't. 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 [16]) – that is for information purposes only, and supposed to support implementations on top of DRMAA.

3.8.3 File Transfer Specifications

The syntax of a file transfer directive for the job description is modeled on the LSF syntax, 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.

3.8.4 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.

3.8.5 Job Identifiers

The job ID 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 job id information, the job id 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 unix 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 job id, 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.

3.8.6 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::attribute
                        // 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: 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: JobInteractive
// 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 and 'IncorrectParameter' error with an
//         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
//
```

```
// 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
//
// 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
//
// 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
//
// 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
//       - syntax similar to LSF (see earlier notes)
//       - available in JSDL, DRMAA
//
```

```
// name: Cleanup
// desc: defines if output files get removed after 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 finishes
//         - On 'True', output files MUST be deleted
//           after job 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.
//         - 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: WallclockSoftLimit
// desc: estimate of wall clock time in seconds which
//       job will require. This attribute is
// mode: ReadWrite, optional
// type: Int
// value: -
// notes: - intended to provide hints to the scheduler.
//         - if limit is reached, the action taken is
//           specific to the resource manager and its
//           scheduling policies.
//         - available in DRMAA
//         - not supported by 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: 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: 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
//         - availale 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
//         - availale 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
//         - availale in JSDL
//
// name: CandidateHosts
// desc: list of host names which to be considered by
//       the resource manager as candidate targets
// mode: ReadWrite, optional
// type: Vector String
// value: -
// notes: - semantics as defined by JSDL
//         - availale 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 "queue", many back end systems
//         can make use of this attribute.
//         - not supported by JSDL
}

class job_service : implements saga::object
                  implements saga::async
                  // from object saga::error_handler
{
    CONSTRUCTOR      (in session      session,
                     in string      rm = "",
                     out job_service service);
    DESTRUCTOR       (in job_service  service);
}

```

```

    create_job      (in  job_description job_desc,
                    out  job              job);
    run_job         (in  string            host = "",
                    in  string            cmdline,
                    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::attribute
             // from task  saga::object
             // from task  saga::monitorable
             // from object saga::error_handler
{
    CONSTRUCTOR      (void                );
    DESTRUCTOR       (in  job              job);

    // job inspection
    get_job_description (out job_description job_desc);
    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  job_desc);
    signal            (in  int              signum);

    // Attributes:
    //  name:  JobID
    //  desc:  SAGA representation of the job identifier
    //  mode:  Read
    //  type:  String
    //  value: -
    //  notes: - format: as described earlier
}

```

```
//
// name: ExecutionHosts
// desc: list of host names or IP addresses allocated
//       to run this job
// mode: Read, optional
// type: Vector String
// value: -
// notes: -
//
// name: Created
// desc: time stamp of the job creation in the
//       resource manager
// mode: Read, optional
// type: Time
// value: -
// notes: - can be interpreted as submission time
//
// name: Started
// desc: time stamp indicating when the job started
//       running
// mode: Read, optional
// type: Time
// value: -
//
// name: Finished
// desc: time stamp indicating when the job completed
// mode: Read, optional
// type: Time
// value: -
//
// name: WorkingDirectory
// desc: working directory on the execution host
// mode: Read, 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: Read, 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: Read, optional
//      type: Int
//      value: -
//      notes: - only available in final states, if at all

// Metrics:
//      name: State
//      desc: fires on state changes of the job, and has
//            the literal value of the job state enum.
//      mode: Read
//      unit: 1
//      type: Enum
//      value: "Unknown"
//      notes: - the state metric is inherited from
//            saga::task, but has a different set
//            of possible values
//            - see description of job states above
//
//      name: StateDetail
//      desc: fires as a job changes its state detail
//      mode: Read, optional
//      unit: 1
//      type: String
//      value: -
//
//      name: Signal
//      desc: fires as a job receives a signal, and has a
//            value indicating the signal number
//      mode: Read, optional
//      unit: 1
//      type: Int
//      value: -
//      notes: - no guarantees are made that any or all
//            signals can be notified by this metric
//
//      name: CPUTimeLimit
//      desc: number of cpu seconds consumed by the job
//      mode: Read, optional
//      unit: seconds
//      type: Int
```

```
// value: -
// notes: - aggregated across all processes/threads
//
// name: MemoryUse
// desc: current aggregate memory usage
// mode: Read, optional
// unit: megabyte
// type: Float
// value: "0.0"
// notes: - metric becomes 'Final' after Job completions,
//         and then shows the memory high water mark
//
// name: VmemoryUse
// desc: current aggregate virtual memory usage
// mode: Read, optional
// unit: megabyte
// type: Float
// value: "0.0"
// notes: - metric becomes 'Final' after Job
//         completions, and then shows the virtual
//         memory high water mark
//
// name: Performance
// desc: current performance
// mode: Read, optional
// unit: FLOPS
// type: Float
// value: "0.0"
// notes: - metric becomes 'Final' after Job
//         completions, and then shows the performance
//         high water mark
}

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

```
}
```

3.8.7 Details

```
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 'Attribute' 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 back end systems without any further specification.

There should be much 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: -
Outputs: obj: the newly created object
Throws: NotImplemented
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
Outputs: -
Throws: -

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 - that probably implies that implementations need to internally track what resource manager (or job_service) created the job.

- CONSTRUCTOR

Purpose: create the object

Format: CONSTRUCTOR (in session session,
in string rm = "",
out job_service obj)

Inputs: session: session to associate with
the object
rm: contact string for resource
manager
Outputs: obj: the newly created object

Throws: NotImplemented
IncorrectURL
AuthenticationFailed
AuthorizationFailed
PermissionDenied
DoesNotExist
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 given for the rm cannot be parsed by the implementation, a '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
Outputs: -
Throws: -

- create_job

Purpose: create a job instance
Format: create_job (in job_description job_desc,
out job job);
Inputs: job_desc: description of job to be
submitted
Outputs: job: a job object representing
the submitted job instance
Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
Timeout
NoSuccess
Throws: BadParameter
PreCon: - job_des MUST have a valid 'Executable'
attribute, otherwise a 'BadParameter'
exception is thrown.
PostCond: - the returned job is in the New state
- the job_description is deep_copied (no state
is shared after the method invocation)
Notes: - calling run() on the job will submit it to
the resource, and advance its state.
- 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 host = "",
in string commandline,
out job job,
out opaque stdin,
out opaque stdout,
out opaque stderr);

Inputs: `host:` hostname to be used by rm for
submission
`commandline:` the command and arguments
to be run

Outputs: `stdin:` IO handle for the running
jobs standard input stream
`stdout:` IO handle for the running
jobs standard output
`stderr:` IO handle for the running
jobs standard error
`job:` a job object representing
the submitted job instance

Throws: `NotImplemented`
`AuthenticationFailed`
`AuthorizationFailed`
`PermissionDenied`
`BadParameter`
`IncorrectState`
`Timeout`
`NoSuccess`

PostCond: - the returned job is in the 'Running' state

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 `commandline`, 'JobInteractive' 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 will choose an arbitrary host for execution.
-
- list
 - Purpose: Get a list of jobs which are currently known by the resource manager.
 - Format: list (out array<string> job_ids);
 - Inputs: -
 - Outputs: job_ids: an array of job identifiers
 - Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
Timeout
NoSuccess
 - Notes:
 - The semantics of which jobs are viewable by the calling user context, or how long a resource manager keeps job information, are 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).

 - 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
 - Outputs: job: a job object representing the job identified by job_id
 - Throws: NotImplemented

AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
DoesNotExist
Timeout
NoSuccess

PostCond: - Multiple job instances returned by calling this method with the same argument do not share state (but usually will reflect the same state).

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: -

Outputs: self: a job_self object representing `_this_job`.

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
Timeout
NoSuccess

PostCond: - the returned job_self is, by definition, in 'Running' state.
- instances returned by multiple invocations of this method do not share state (although may reflect the same state).

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 an 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 effect an action to be taken. The API implementation is designed to be agnostic of the back end implementation, such that any back end 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 'Attribute' 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 runt time attributes.

- CONSTRUCTOR

Purpose: create the object

Format: CONSTRUCTOR (out job obj);

Inputs: -

Outputs: obj: the newly created object

Throws: NotImplemented

PostCond: - the returned job is in 'Unknown' state

Notes: - the constructor serves only the purpose to create jobs to be passed by reference to asynchronous `create_job` method of the `job_service` class.

- if any method is called on the created job before it was initilized by a 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
 - Outputs: -
 - Throws: -
 - Notes: - the object destruction does not imply a cancel() on the job.

- 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: -
 - Outputs: jd: a job_description object
 - PreCond: - the job can be in any state
 - PostCond: - the returned job_description is a deep copy
 - Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
IncorrectState
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 'NoSuccess' exception is thrown, with an descriptive error message.
- if this method is called on a job in 'Unkown' state, an 'IncorrectState' exception is thrown.

- get_stdin
 - Purpose: retrieve input stream for a job.
 - Format: get_stdin (out opaque stdin)
 - Inputs: -
 - Outputs: stdin: standard input stream for

the job

PreCond: - the job was submitted via `run_job()`, or with a `job_description` which had the attribute `'JobInteractive'` set to `'True'` - otherwise a `'IncorrectState'` error is thrown.

Throws: `NotImplemented`
`AuthenticationFailed`
`AuthorizationFailed`
`PermissionDenied`
`BadParameter`
`IncorrectState`
`DoesNotExist`
`Timeout`
`NoSuccess`

Notes: - if preconditions are met, but the standard input stream is not available for some reason, a `'DoesNotExist'` exception is thrown.

- `get_stdout`

Purpose: retrieve output stream of job

Format: `get_stdout` (out opaque stdout)

Inputs: -

Outputs: `stdout:` standard output stream for the job

Throws: `NotImplemented`
`AuthenticationFailed`
`AuthorizationFailed`
`PermissionDenied`
`BadParameter`
`IncorrectState`
`DoesNotExist`
`Timeout`
`NoSuccess`

PreCond: - the job was submitted via `run_job()`, or with a `job_description` which had the attribute `'JobInteractive'` set to `'True'` - otherwise a `'IncorrectState'` error is thrown.

Notes: - if preconditions are met, but the standard output stream is not available for some reason, a `'DoesNotExist'` exception is thrown.

- `get_stderr`

Purpose: retrieve error stream of job

Format: `get_stderr` (out opaque stderr)

Inputs: -
Outputs: stderr: standard error stream for
the job
Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
DoesNotExist
Timeout
NoSuccess
PreCond: - the job was submitted via run_job(), or with
a job_description which had the attribute
'JobInteractive' set to 'True' - otherwise
a 'IncorrectState' error is thrown.
Notes: - if preconditions are met, but the standard
error stream is not available for some
reason, a 'DoesNotExist' exception is thrown.

Job Management Methods:

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

- resume
Purpose: Ask the resource manager to perform a resume

operation on a suspended job.

Format: resume (void);

Inputs: -

Outputs: -

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
IncorrectState
Timeout
NoSuccess

PreCond: - the job must be in 'Suspended' state

PostCond: - on success, the job is in 'Running' state
- on failure, the job is in 'Suspended' state

Notes: - if the job is not in 'Suspended' state, a
'IncorrectState' exception is thrown.

- checkpoint

Purpose: Ask the resource manager to initiate a checkpoint operation on a running job.

Format: checkpoint (void);

Inputs: -

Outputs: -

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
IncorrectState
Timeout
NoSuccess

PreCond: - the job must be in 'Running' state

PostCond: - the job is in 'Running' state

Notes: - The semantics of checkpoint, and the actions taken to initiate a checkpoint, are resource manager specific. In particular, the implementation/backend can trigger either a system level or an application level
- if the job is not in 'Running' state, a
'IncorrectState' exception is thrown.
checkpoint.

- migrate

Purpose: Ask the resource manager to migrate a job.

Format: migrate (in job_description job_desc);

```

Inputs:  job_desc:          new job parameters to apply
                               when the job is migrated

Outputs: -

Throws:  NotImplemented
         AuthenticationFailed
         AuthorizationFailed
         PermissionDenied
         BadParameter
         IncorrectState
         Timeout
         NoSuccess

PreCond: - the job must be in 'Running' or 'Suspended'
          state

PostCond: - the job keeps its state.
          - the job_description does not share state with
            the job or other saga objects - it is deep
            copied.

Notes:   - job_desc 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. That is not a
            condition though.
          - if the job is not in 'Running' or 'Suspended'
            state, a '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

  Outputs: -

  Throws:  NotImplemented
         AuthenticationFailed
         AuthorizationFailed
         PermissionDenied
         BadParameter
         IncorrectState

```

```

        Timeout
        NoSuccess
    PreCond: - the job must be in 'Running' or 'Suspended'
              state
    PostCond: - the job can remain in its state, or can go to
              'Running', 'Suspended', or any final state.
    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 that the signal number is not supported by
              the backend, a 'BadParameter' exception is
              thrown.
              - if the job is not in 'Running' or 'Suspended'
              state, a '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 be 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 obj);
 Inputs: -
 Outputs: obj: the newly created object
 Throws: NotImplemented
 PostCond: - the returned job_self is in 'Unknown' state
 Notes: - the constructor serves only the purpose to
 create jobs to be passed by reference to
 asynchronous get_self method of the
 job_service class.
 - if any method is called on the created
 job_self before it was initialized by a
 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
 Outputs: -
 Throws: -
 Notes: - destruction of job_self does not imply a
 cancel() on the application.

3.8.8 Examples

Code Example

```

1  Example : simple job submission and polling for finish.
2
3  // -----
4  // c++ example
5  std::list <string>  transfers;
6  saga::job_description jobdef;
7
8  transfers.push_back ("infile > infile");
9  transfers.push_back ("ftp://host.net/path/out << outfile");
10
11  jobdef.set_attribute      ("'Executable'",   "job.sh");
12  jobdef.set_attribute      ("'TotalCPUCount'", "16");
13  jobdef.set_vector_attribute ("'FileTransfer'", transfers);
14
15  saga::job_service js;
16  saga::job          job = js.create_job ("remote.host.net",
17                                         jobdef);
  
```

```
18 job.run ();
19
20 while ( 1 )
21 {
22     // get job state
23     saga::job::state state = job.get_state ();
24
25     // get list of hosts the job is/where running on
26     std::list <std::string> hostlist = job.get_attribute
27         ("ExecutionHosts");
28
29     if ( saga::job::Running == state )
30     {
31         std::cout << "Job is running." << std::endl;
32     }
33     else if ( saga::job::Suspended == state )
34     {
35         std::cout << "Job is suspended." << std::endl;
36     }
37     else if ( saga::job::Done == state )
38     {
39         std::cout << "Job completed successfully." << std::endl;
40         exit (0);
41     }
42     else if ( saga::job::Canceled == state )
43     {
44         // this should never occur, as cancel is not called.
45         std::cout << "Job canceled." << std::endl;
46         exit (1);
47     }
48     else
49     {
50         // state can only be 'Failed'
51         assert(saga::job::Failed == state);
52
53         std::string exitcode = job.get_attribute ("ExitCode");
54
55         std::cout << "Job failed with " << exitcode << std::endl;
56         exit (exitcode);
57     }
58
59     sleep (1); // idle
60 }
```

3.9 SAGA Name Spaces

Several SAGA packages share the notion of namespaces and operations on these namespaces. In order to increase consistency in the API, those packages share the same API paradigms. This section describes those paradigms, and those classes which operate on arbitrary hierarchical namespaces, 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 (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 namespace per se, but belong to the specific implementations, e.g. physical files, which implement the namespace interfaces.

3.9.1 Definitions

The Grid File System Working Group in OGF has defined a Resource Namespace Service (RNS [15]). The SAGA Core API Specification follows the definition of a namespace from that document.

Directory: A 'Directory' represents what [15] defines as 'Virtual Directory':

“A virtual directory is an RNS entry that is represented as a non-leaf node in the hierarchical namespace tree. When rendered by a namespace 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 namespace. A virtual directory, therefore, is purely a namespace 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 namespace 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 namespace and strictly serve to facilitate

hierarchy. Namespace 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 [15] 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 namespace. 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 namespace 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 namespace junction within a valid WS- Addressing [2] 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 [15] (formatting changed):

“Entry names are composed of a simple string of human readable characters. Since certain characters serve special purposes both within the namespace 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 (/).”

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, pathnames 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 `cwd` of the object instance the operation is performed upon, e.g. when they start with `'./'` or `'../'`).

Note that the comments from the Introduction, subsection 2.11, apply here. In particular, an implementation MAY throw an `IncorrectURL` exception if it is unable to handle a given pathname.

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 an URL pointing to the link target.

At the moement, [15] does not have a notion of symbolic links. However, an RNS 'junction' which is associated with another RNS junction can be regarded as symbolic link.

Wildcards: The API supports wildcards where appropriate, and thereby follows the POSIX standard 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 for more details. In the 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 dependent 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 ambiguity may arise.

For the method calls on `saga::ns_entry`, NO wildcards are allowed. 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). Flags MUST be applied to all elements of a wildcard expansion, even if that raises an exception for any reasons.

Access Control Lists – ACLs: ACLs are adopted to express access permissions. As of now it is somewhat unclear on what subjects should ACLs operate in grid environments: user id's? distinguished names? groups? This document settles for distinguished names but additionally allows a '*' wildcard for `set_acl()`, which enables to set ACLs for more than one user, or 'groups':

```
dn_user = "0=dutchgrid, 0=vu, CN=Joe Doe";
dn_group = "0=dutchgrid, 0=vu, CN=*";
dn_group = "0=dutchgrid, 0=project-123, CN=*";
```

```
dn_group = "O=*, O=project-123, CN=";
```

An implementation MAY raise an `InvalidParameter` exception if wildcards in ACL specifications are not supported – this MUST be documented by the implementation.

Queries for ACLs (`get_acl()`), are supposed to be performed for an individual DN, not a group of DN's (e.g. the DN should not contain a `*`). An implementation MAY support queries for patterns, but MUST then return the most restrictive set of ACLs available for any single DN matching the pattern.

If name space entities are newly created, they inherit the ACL's of the name space directory they are created in. However, new file entries (i.e. non-directory entries) get the executable ACL stripped off. If entries get moved, copied or linked into a new location, they maintain the original set of ACLs, and in particular stay executable.

We are well aware that this approach is somewhat arbitrary – no other suitable approach is, however, known to us, in the scope of current Grid standardizations. We intent to review this approach as soon as some standard emerges in that area.

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 name space entry instance was closed, all method calls on that instance 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 corrupted 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 occurred 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.

3.9.2 Specification

```
package saga.name_space
{
  enum flags
  {
    None           = 0,
    Overwrite      = 1,
    Recursive      = 2,
    Dereference    = 4,
    Create         = 8,
    Excl           = 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
                 // from object saga::error_handler
  {
    CONSTRUCTOR    (in session      session,
                   in string       url,
                   in int          flags = None);

    DESTRUCTOR     (out ns_entry    obj   );
                   (in ns_entry    obj   );

    // basic properties
    get_url        (out string      url   );
    get_cwd        (out string      cwd   );
    get_name       (out string      name  );
  }
}
```

```

// navigation/query methods
is_dir      (in int          flags = None,
             out boolean     test  );
is_entry    (in int          flags = None,
             out boolean     test  );
is_link     (in int          flags = None,
             out boolean     test  );
read_link   (out string      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 string       target,
             in int          flags = None);
link        (in string       target,
             in int          flags = None);
move        (in string       target,
             in int          flags = None);
remove      (in int          flags = None);
close       (in float        timeout = 0.0);
}

class ns_directory : extends    saga::ns_entry
                        // from ns_entry saga::object
                        // from ns_entry saga::async
                        // from object  saga::error_handler
{
    CONSTRUCTOR    (in session    session,
                   in string     url,
                   in int        flags = None,
                   out ns_directory obj  );
    DESTRUCTOR     (in ns_directory obj  );

    // navigation/query methods
    change_dir     (in string     dir   );
    list           (in string     pattern = "",
                   out array<string> names );
    find           (in string     pattern,

```

```

        in int          flags = Recursive,
        out array<string> names );
read_link    (in string   name,
              out string  link   );
exists      (in string   name,
              out boolean exists );
is_dir      (in string   name,
              in int     flags = None,
              out boolean test  );
is_entry    (in string   name,
              in int     flags = None,
              out boolean test  );
is_link     (in string   name,
              in int     flags = None,
              out boolean test  );

// manage entries by number
get_num_entries (out int     num   );
get_entry      (in int     entry,
               out string  name  );

// security
set_acl       (in string   name,
               in string   dn,
               in int     acl,
               in int     flags = None);
get_acl       (in string   name,
               in int     flags = None,
               out int    acl   );
list_dn      (in string   name,
               in int     flags = None,
               out array<string> dn   );

// management methods
copy         (in string   source,
              in string   target,
              in int     flags = None);
link        (in string   source,
              in string   target,
              in int     flags = None);
move        (in string   source,
              in string   target,
              in int     flags = None);
remove      (in string   target,
              in int     flags = None);
make_dir    (in string   target,
```

```

                                in int          flags = None);

    // factory methods
    open      (in string      name,
              in int         flags = None,
              out ns_entry   entry );
    open_dir  (in string      name,
              in int         flags = None,
              out ns_directory dir  );
}
}

```

3.9.3 Details

```
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 entries 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. The URL returned on `get_url()` should serve as base for the return values on `get_cwd()` and `get_name()`: for directory type entries, `get_url()` and `get_cwd()` MUST return identical URLs. For not-directory type entries, the URL returned on `get_url` MUST equal the concatenation of the return values of `get_cwd()` and `get_name()`.

Constructor / Destructor:

```

- CONSTRUCTOR
  Purpose: create the object
  Format:  CONSTRUCTOR      (in session session,
                           in string url,
                           in int   flags = None,
                           out ns_directory obj)

  Inputs:  session:      session handle
          url:           initial working dir
          flags:         open mode

  Outputs: obj:          the newly created object

  Throws:  NotImplemented
          IncorrectURL
          AuthenticationFailed
          AuthorizationFailed
          PermissionDenied
          BadParameter
          DoesNotExist
          Timeout
          NoSuccess

  Notes:  - the default flag set is 'None' (0)
          - the constructor performs an open of the
            entry - all notes to the respective open
            call apply.

- DESTRUCTOR
  Purpose: destroy the object
  Format:  DESTRUCTOR      (in ns_entry   obj)
  Inputs:  obj:            the object to destroy
  Outputs: -
  PostCond: - the instance is closed.
  Throws:  -
  Notes:  - if the instance was not closed before, the
            destructor performs a close() of the
            entry, 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 string url);
  Inputs:  -

```

Outputs: url url pointing to the entry
Throws: NotImplemented
IncorrectState
Timeout
NoSuccess
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 string cwd);
Inputs: -
Outputs: cwd current working directory
Throws: NotImplemented
IncorrectState
Timeout
NoSuccess
Notes: - if the instance was not opened before, an
'IncorrectState' exception is thrown.

- get_name
Purpose: obtain the name part of the url
Format: get_name (out string name);
Inputs: -
Outputs: name last part of the pathname
Throws: NotImplemented
IncorrectState
Timeout
NoSuccess
Notes: - if the instance was not opened before, an
'IncorrectState' exception is thrown.

- is_dir
Purpose: tests entry for beeing a directory
Format: i s_dir (in int flags = None,
out boolean test);
Inputs: flags: flags for operation
Outputs: test: boolean indicating if entry
is a directory

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
Timeout
NoSuccess

Notes: - returns true if entry is a directory, false otherwise
- flag can be set to 'Dereference', default is 'None' (0).
- other flags are not allowed on this method, and cause an '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 entry for being a ns_entry

Format: is_entry (in int flags = None,
out boolean test);

Inputs: flags: flags for operation

Outputs: test: boolean indicating if entry
is a ns_entry

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
Timeout
NoSuccess

Notes: - the method returns false if the entry is a link or a directory (although a ns_directory IS_A ns_entry, false is returned on a test on a ns_directory) - otherwise true is returned.
- flag can be set to 'Dereference', default is 'None' (0)
- other flags are not allowed on this method, and cause an '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 beeing a link

Format: `is_link` (in int flags = None,
out boolean test);

Inputs: flags: flags for operation

Outputs: test: boolean indicating if
entry is a link

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
Timeout
NoSuccess

Notes: - returns true if the entry is a link, false
otherwise
- flag can be set to 'Dereference', default is
'None' (0)
- other flags are not allowed on this method,
and cause an '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 string link);

Inputs: -

Outputs: link: resolved name

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
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, 'BadParameter'
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
flags: flags defining the operation
modus

Outputs: -

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
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 flag set is '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
 - Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
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 flag set is '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

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
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 flag set is '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 namespace

Format: copy (in string target,
in int flags = None);

Inputs: target: name to copy to
flags: flags defining the operation
modus

Outputs: -

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
AlreadyExists
Timeout
NoSuccess

- Notes:
- if the target is a directory the source entry is copied into that directory
 - it is a 'BadParameter' error if the source is a directory and the 'Recursive' flag is not set.
 - it is a 'BadParameter' error 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, an 'DoesNotExist' error 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' error.
 - if the instance points at an symbolic link, the source is deeply dereferenced before copy. If dereferencing is impossible (e.g. on a broken link), an 'InvalidState' exception is thrown.
 - other flags are not allowed, and cause a 'BadParameter' exception.
 - the default flags set is 'None' (0).
 - similar to 'cp' as defined by POSIX
- link
- Purpose: create a symbolic link from the entry to the target entry
- Format: link (in string target,
in int flags = None);
- Inputs: target: name to link to
flags: flags defining the operation
modus
- Outputs: -
- Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
AlreadyExists
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.
 - it is a 'BadParameter' error 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, an 'DoesNotExist' error 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' error.
 - if the instance points at an symbolic link, the source is not dereferenced before linking, unless the 'Dereference' flag is given. If dereferencing is impossible (e.g. on a broken link), an 'InvalidState' exception is thrown.
 - other flags are not allowed, and cause a 'BadParameter' exception.
 - the default flags set is 'None' (0).
 - similar to 'ln' as defined by POSIX
- move
- Purpose: rename source to target, or move source to target if target is an directory.
- Format: move (in string target,
in int flags = None);
- Inputs: target: name to move to
flags: flags defining the operation
modus
- Outputs: -
- Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
AlreadyExists

Timeout
NoSuccess

Notes:

- if the target is a directory the source entry is moved into that directory
- it is a 'BadParameter' error if the source is a directory and the 'Recursive' flag is not set.
- it is a 'BadParameter' error 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, an 'DoesNotExist' error 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' error.
- if the instance points at an symbolic link, the source is not dereferenced before moving, unless the 'Dereference' flag is given. If dereferencing is impossible (e.g. on a broken link), an 'InvalidState' exception is thrown.
- other flags are not allowed, and cause a 'BadParameter' exception.
- the default flags set is '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

Outputs: -

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
Timeout
NoSuccess

Notes:

- it is a 'BadParameter' error if the source is a directory and the 'Recursive' flag is not set.
- it is a 'BadParameter' error 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 dereferencing is impossible (e.g. on a broken link), an 'InvalidState' exception is thrown.
 - other flags are not allowed, and cause a 'BadParameter' exception.
 - the default flags set is 'None' (0).
 - the method implies a call on close(), and all side effects from 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
- Outputs: -
- Throws: NotImplemented
IncorrectState
NoSuccess
- Notes:
- 'IncorrectState' is thrown if the object was closed or removed before.
 - any subsequent method call on the object MUST also raise 'IncorrectState' (apart from DESTRUCTOR).
 - 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 Introduction.
 - for timeout semantics, see Introduction.

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 session,
in string url,
in int flags = None,
out ns_directory obj)

Inputs: url: initial working dir
flags: open mode
session: session handle for
object creation

Outputs: obj: the newly created object

Throws: NotImplemented
IncorrectURL
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
DoesNotExist
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 flag set is 'None' (0).

- DESTRUCTOR

Purpose: destroy the object

Format: DESTRUCTOR (in ns_directory obj)

Inputs: obj: the object to destroy

Outputs: -

Throws: -

Notes: - the semantics of the inherited destructors
apply

Methods for navigation in the namespace hierarchy:

- change_dir

Purpose: change the working directory
Format: change_dir (in string dir);
Inputs: dir: directory to change to
Outputs: -
Throws: NotImplemented

IncorrectURL
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
DoesNotExist
Timeout
NoSuccess

Notes: - if 'dir' can be parsed as URL, but contains an
invalid directory name, an 'BadParameter'
exception is thrown.
- if 'dir' does not exist, a 'DoesNotExist'
exception is thrown.
- similar to the 'cd' command in Unix shells,
as defined by POSIX

- list

Purpose: list entries in this directory
Format: list (in string pattern = "",
out array<string> names);
Inputs: pattern: name or pattern to list
Outputs: names: array of names matching the
pattern

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
Timeout
NoSuccess

Notes:

- if pattern is not given (i.e. empty string), all entries in the current working directory are listed.
- if the pattern cannot be parsed, a 'BadParameter' exception with an descriptive error message is thrown.
- if the 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 pattern,
in int flags = Recursive,
out array<string> names);

Inputs: pattern: pattern for names of
entries to be found

flags: flags defining the operation
modus

Outputs: names: array of names matching the
pattern

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
Timeout
NoSuccess

Notes:

- the 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 matching name.
- the default flag set is 'Recursive' (1).
- other flags are not allowed, and cause a 'BadParameter' exception.
- the pattern follows the standard unix shell wildcard specification, as described above.
- the matching entries returned are relative (to cwd) path names.
- similar to 'find' as defined by POSIX, but

limited to the `-name` option.

- `read_link`

Purpose: returns the name of the link target

Format: `read_link` (in string name,
out string link);

Inputs: `name:` name to be resolved

Outputs: `link:` resolved name

Throws: `NotImplemented`

`IncorrectURL`

`AuthenticationFailed`

`AuthorizationFailed`

`PermissionDenied`

`BadParameter`

`IncorrectState`

`DoesNotExist`

`Timeout`

`NoSuccess`

Notes:

- all notes to `ns_entry::read_link()` apply
- if 'name' can be parsed as URL, but contains an invalid entry name, an 'BadParameter' exception is thrown.
- if 'name' does not exist, a 'DoesNotExist' exception is thrown.

- `exists`

Purpose: returns true if entry exists, false otherwise

Format: `exists` (in string name,
out boolean exists);

Inputs: `name:` name to be tested for
existence

Outputs: `exists:` boolean indicating existence
of name

Throws: `NotImplemented`

`IncorrectURL`

`AuthenticationFailed`

`AuthorizationFailed`

`PermissionDenied`

`BadParameter`

`IncorrectState`

`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 string name,
in int flags = None,
out boolean test);

Inputs: name: name to be tested
flags: flags for operation

Outputs: test: boolean indicating if name
is a directory

Throws: NotImplemented
IncorrectURL
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
DoesNotExist
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 flag set is '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, an '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

- is_entry

Purpose: tests name for being a ns_entry

Format: is_entry (in string name,

```

                                in int    flags = None,
                                out boolean test);
Inputs:  name:                  name to be tested
         flags:                 flags for operation
Outputs: test:                 boolean indicating if name
                                is a non-directory entry
Throws:  NotImplemented
         IncorrectURL
         AuthenticationFailed
         AuthorizationFailed
         PermissionDenied
         BadParameter
         IncorrectState
         DoesNotExist
         Timeout
         NoSuccess
Notes:   - all notes to the ns_ntry::is_entry() method
         apply.
         - the default flag set is '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, an '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

- is_link
Purpose: tests name for beeing a symbolic link
Format:  is_link                (in string name,
                                in int    flags = None,
                                out boolean test);
Inputs:  name:                  name to be tested
         flags:                 flags for operation
Outputs: test:                 boolean indicating if name
                                is a link
Throws:  NotImplemented
         IncorrectURL
         AuthenticationFailed
         AuthorizationFailed
         PermissionDenied

```

BadParameter
 IncorrectState
 DoesNotExist
 Timeout
 NoSuccess

- Notes:
- all notes to the `ns_ntry::is_link()` method apply.
 - the default flag set is '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, an '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

Iterate over large directories:

-
- `get_num_entries`

Purpose: gives the number of entries in the directory

Format: `get_num_entries (out int num);`

Inputs: -

Outputs: num: number of entries in the directory

Throws: NotImplemented
 AuthenticationFailed
 AuthorizationFailed
 PermissionDenied
 IncorrectState
 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 string name);
 Inputs: entry: index of entry to get
 Outputs: name: name of entry at index
 Throws: NotImplemented
 AuthenticationFailed
 AuthorizationFailed
 PermissionDenied
 IncorrectState
 DoesNotExist
 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: -
 Throws: NotImplemented
 IncorrectURL
 AuthenticationFailed

```

        AuthorizationFailed
        PermissionDenied
        BadParameter
        IncorrectState
        DoesNotExist
        Timeout
        NoSuccess
Notes:   - all notes to the ns_entry::set_acl() method
        apply.
        - the default flag set is '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, an '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
Throws:  NotImplemented
          IncorrectURL
          AuthenticationFailed
          AuthorizationFailed
          PermissionDenied
          BadParameter
          IncorrectState
          DoesNotExist
          Timeout
          NoSuccess
Notes:   - all notes to the ns_entry::get_acl() method
        apply.
        - the default flag set is '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, an '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 for operation
           flags:          flags defining the operation
  Outputs: dn:             list of DNs for which ACLs
                           are set on the entry
  Throws:  NotImplemented
           IncorrectURL
           AuthenticationFailed
           AuthorizationFailed
           PermissionDenied
           BadParameter
           IncorrectState
           DoesNotExist
           Timeout
           NoSuccess
  Notes:   - all notes to the ns_entry::get_dn() method
           apply.
           - the default flag set is '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, an 'DoesNotExist' exception is
             thrown.

```

Management of namespace entries:

```

- copy
  Purpose: copy the entry to another part of the namespace
  Format:  copy          (in string source,
                        in string target,
                        in int   flags = None);
  Inputs:  source:      name to copy
           target:      name to copy to

```

```

        flags:                flags defining the operation
                               modus
Outputs:  -
Throws:   NotImplemented
          AuthenticationFailed
          AuthorizationFailed
          PermissionDenied
          BadParameter
          IncorrectState
          AlreadyExists
          DoesNotExist
          Timeout
          NoSuccess
Notes:    - all notes to the ns_entry::copy() method
          apply.
          - the default flag set is '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, an 'DoesNotExist' exception is
            thrown.

- link
  Purpose: create a symbolic link from the source entry to
           the target entry so that any reference to the
           target refers to the source entry
  Format:  link                (in string source,
                               in string target,
                               in int    flags = None);
  Inputs:  source:            name to link
           target:           name to link to
           flags:            flags defining the operation
                               modus
Outputs:  -
Throws:   NotImplemented
          AuthenticationFailed
          AuthorizationFailed
          PermissionDenied
          BadParameter
          IncorrectState
          AlreadyExists
          DoesNotExist
          Timeout
          NoSuccess

```

Notes: - all notes to the `ns_entry::link()` method apply.
- the default flag set is '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, an 'DoesNotExist' exception is thrown.

- move

Purpose: rename source to target, or move source to target if target is an directory.

Format: move (in string source,
in string target,
in int flags = None);

Inputs: source: name to move
target: name to move to
flags: flags defining the operation
modus

Outputs: -

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
AlreadyExists
DoesNotExist
Timeout
NoSuccess

Notes: - all notes to the `ns_entry::move()` method apply.
- the default flag set is '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, an 'DoesNotExist' exception is thrown.
- moving any parent or the current directory (e.g. '.', '..' etc.) is not allowed, and throws a 'BadParameter' exception

```
- remove
Purpose: removes the entry
Format:  remove          (in string target,
                        in int   flags = None);
Inputs:  target:        entry to be removed
Outputs: -
Throws:  NotImplemented
        AuthenticationFailed
        AuthorizationFailed
        PermissionDenied
        BadParameter
        IncorrectState
        AlreadyExists
        DoesNotExist
        Timeout
        NoSuccess
Notes:   - all notes to the ns_entry::remove() method
        apply.
        - the default flag set is '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, an 'DoesNotExist' exception is
          thrown.
        - removing any parent or the current directory
          (e.g. '.', '..' etc.) is not allowed, and
          throws a 'BadParameter' exception

- make_dir
Purpose: creates a new directory
Format:  make_dir       (in string target,
                        in int   flags = None);
Inputs:  target:        directory to create
Outputs: -
Throws:  NotImplemented
        IncorrectURL
        AuthenticationFailed
        AuthorizationFailed
        PermissionDenied
        BadParameter
        IncorrectState
        AlreadyExists
        DoesNotExist
        Timeout
```

```

Notes:      NoSuccess
            - if the parent directory or directories do not
              exist, 'CreateParents' flag must be set or an
              'DoesNotExist' exception will be thrown.
              If set, the parent directories are created as
              well.
            - an 'AlreadyExists' exception is thrown if the
              directory already exists and the 'Excl' flag is
              given.
            - the default flag set is 'None' (0).
            - other flags are not allowed on this method,
              and cause an 'BadParameter' exception.
            - if 'target' can be parsed as URL, but contains
              an invalid entry name, an '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 string name,
                          in int    flags = None,
                          out ns_directory dir);
  Inputs:   name:        directory to open
            flags:       flags defining the operation
                          modus
  Outputs:  dir:         opened directory instance
  PostCond: - the session of the returned 'dir' is that of
            the calling ns_directory instance.
  Throws:   NotImplemented
            IncorrectURL
            IncorrectSession
            AuthenticationFailed
            AuthorizationFailed
            PermissionDenied
            BadParameter
            IncorrectState
            AlreadyExists
            DoesNotExist
            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 'Creat' flag is not
              given.

```

- a 'AlreadyExist' exception is thrown if 'name' does exist and the 'Creat' flag and the 'Excl' flag are given.
 - no exception is thrown if 'name' does exist and the 'Create' flag is given, and the 'Excl' flag is not given.
 - if the 'Creat' flag is given, all notes to the ns_directory::make_dir() method apply.
 - the default flag set is 'None' (0).
 - other flags are not allowed on this method, and cause an 'BadParameter' exception.
 - 'name' is always deeply dereferenced, however, the cwd is still set to 'name', and not to the value of the link target.
 - if 'name' can be parsed as URL, but contains an invalid directory name, an '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 string name,
in int flags = None,
out ns_entry entry);
- Inputs: name: entry
flags: flags defining the operation
modus
- Outputs: entry: opened entry instance
- PostCond: - the session of the returned 'file' is that of the calling ns_directory instance.
- Throws: NotImplemented
IncorrectURL
IncorrectSession
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState

AlreadyExists

DoesNotExist

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 'Creat' flag is not given.
 - a 'AlreadyExists' exception is thrown if 'name' does exist, and the 'Creat' and 'Excl' flags are given.
 - 'name' is always deeply dereferenced, the cwd, however, is not changed to the link targets cwd.
 - the file is locked on open if the 'Lock' flag is given. If the file is already in a locked state, the open will fail and a descriptive error will be issued. If a file is opened in locked mode, any other open on that file MUST fail with a 'NoSuccess' exception if the 'Lock' flag is given. Note that a file can be opened in normal mode, and then in locked mode, w/o an error getting raised. The application programmer must take precautions to avoid such situations. The lock will get removed on destruction of the file object, and also on close. If an implementation does not support locking, an descriptive 'BadParameter' error MUST get thrown if the 'Lock' flag is given.
 - the default flag set is 'None' (0).
 - other flags are not allowed on this method, and cause an 'BadParameter' exception.
 - in the asynchronous case, the resulting entry instance is passed as 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
-

3.9.4 Examples:

Code Example

```
1 More examples are given in the File and Logical_File sections.
2
3 Example: provide recursive directory listing for a given
4 directory
5
6 Note: - check for '.' and '..' recursion are left as an
7 exercise to the reader...
8 - string operations and printf statements are
9 obviously simplified...
10
11 +-----+
12 // c++ example
13 std::string indent (int indent)
14 {
15     std::string s = " ";
16
17     for (int i = 0; i < indent; i++, s += " ");
18
19     return (s);
20 }
21
22 void list_dir (std::string & url,
23               int          indent = 0)
24 {
25     try
26     {
27         // create directory and iterate over entries
28         saga::ns_dir dir (url);
29
30         printf ("\n%s ---> %s\n", indent (indent), url);
31
32         for ( int i = 0; i < dir.get_num_entries (); i++ )
33         {
34             char type = '?';
35             string info = "";
36
37             // get name of next entry
38             string name = dir.get_entry (i);
39
40             // get type and other infos
41             if ( dir.is_link (name) )
42             {
43                 if (dir.exists(dir.read_link (name))){info="---> ";}
44                 else {info="-|-> ";}
45                 info += dir.read_link (name);
46                 type = 'l';
```

```

47     }
48     else if (dir.is_entry(name)){ type = 'f';           }
49     else if (dir.is_dir (name)){ type = 'd'; info = "/";}
50
51     printf ("%s > %3d - %s - %s%s\n",
52            indent (indent), i + 1,
53            type, name, info);
54
55     // recursion on directories
56     if ( dir.is_dir (name) )
57     {
58         list_dir (name, indent++);
59     }
60 }
61
62 printf ("\n%s <--- %s\n", indent (indent), url);
63 }
64
65 // catch all errors - see elsewhere for better examples
66 // of error handling in SAGA
67 catch ( const saga::exception & e )
68 {
69     std::cerr << "Oops! SAGA error: "
70               << e.what () << std::endl;
71 }
72
73 return;
74 }
75
76 +-----+
77
78 // a C++ example for ACL management
79 {
80     // allow short forms of flags
81     using namespace saga::ns_entry;
82
83     std::string dn_user = "O=dutchgrid, O=vu, CN=Andre Merzky";
84     std::string dn_group = "O=dutchgrid, O=vu, CN=*";
85
86     // open file (default: Read only)
87     saga::file f (url);
88
89     // set ACL restrictions for file. The ACL set is
90     // performed with the permissions of the session context
91     f.set_acl (dn_user, ACL_Read | ACL_Write);
92     f.set_acl (dn_group, ACL_Read);
93
94     // check if acl allow write with our current session
95     // contexts
96     if ( f.get_acl () & ACL_Write )

```

```
97     {  
98         saga::file f_2 (url, ReadWrite);  
99  
100        f_2.write ("data", 4);  
101    }  
102 }
```

3.10 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 useful to note that interactions with files as opaque entities (i.e., as entries in file name spaces) are covered by the name space 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 name space package apply if available, unless noted here otherwise.

The described classes are syntactically and semantically POSIX oriented. Large numbers of simple POSIX like remote data access operations are 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' 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 vector versions of the standard POSIX `read()/write()` methods; the arguments are vectors of instructions and buffers to operate on. In other words, `readv()` and `writev()` can be regarded as specialized bulk methods, which cluster multiple I/O operations into a single operation. Advantages 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 occurrence 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 bandwidth 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) [9]. The pattern based I/O routines in SAGA use such descriptions to reduce the bandwidth limitation of scattered I/O. The advantages 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.

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 pattern, 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.

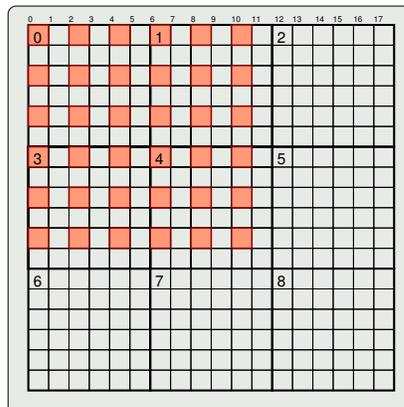


Figure 5: The highlighted elements are defined by "(0,17,36,6,(0,0,2,6))".

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 arbitrary 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 se-

matic 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.

3.10.1 Specification

```
package saga.file
{
  enum flags
  {
    None          = 0, // same as in name_space::flags
    Overwrite     = 1, // same as in name_space::flags
    Recursive     = 2, // same as in name_space::flags
    Dereference   = 4, // same as in name_space::flags
    Create        = 8, // same as in name_space::flags
    Excl         = 16, // same as in name_space::flags
    Lock          = 32, // same as in name_space::flags
    CreateParents = 64, // same as in name_space::flags
    Truncate     = 128,
    Append        = 256,
    Read          = 512,
    Write         = 1024,
    ReadWrite     = 2048,
    Binary        = 4096
  }

  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 directory : extends          saga::ns_directory
                    // from ns_directory saga::ns_entry
                    // from ns_entry    saga::object
                    // from ns_entry    saga::async
                    // from object      saga::error_handler
{
    CONSTRUCTOR (in    session          session,
                 in    string          url,
                 in    int            flags = Read,
                 out   directory      dir    );
    DESTRUCTOR  (in    directory      dir    );

    // inspection methods
    get_size    (in    string          name,
                 in    int            flags = None,
                 out   int            size   );
    is_file     (in    string          name,
                 in    int            flags = None,
                 out   boolean        test   );

    // factory like methods
    open_dir    (in    string          name,
                 in    int            flags = Read,
                 out   directory      dir    );

    open        (in    string          name,
                 in    int            flags = Read,
                 out   file           file   );
}

class 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          session,
                 in    string          url,
                 in    int            flags = Read,
                 out   file           file   );
    DESTRUCTOR  (in    file           file   );
}
```

```
// inspection
get_size  (out  int           size  );

// POSIX like I/O
read      (in  int           len_in,
           inout array<byte> buffer,
           out  int           len_out );
write     (in  int           len_in,
           in   array<byte>  buffer,
           out  int           len_out );
seek      (in  int           offset,
           in   seek_mode    whence,
           out  int           position );

// scattered I/O
read_v    (inout array<ivec>  ivec  );
write_v    (inout array<ivec>  ivec  );

// pattern based I/O
size_p    (in  string        pattern,
           out  int           size  );
read_p    (in  string        pattern,
           inout array<byte>  buffer,
           out  int           len_out );
write_p   (in  string        pattern,
           in   array<byte>  buffer,
           out  int           len_out );

// extended I/O
modes_e   (out  array<string>  emodes );
read_e    (in  string        emode,
           in   string        spec,
           inout array<byte>  buffer,
           out  int           len_out );
write_e   (in  string        emode,
           in   string        spec,
           in   array<byte>  buffer,
           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
  }
}
```

3.10.2 Details

class directory:

- CONSTRUCTOR

Purpose: open the directory

Format: CONSTRUCTOR (in session session,
in string url,
in int flags = Read,
out directory dir)

Inputs: session: session to associate the
object with

url: location of directory

flags: mode for opening

Outputs: dir: the newly created object

Throws: NotImplemented
IncorrectURL
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
AlreadyExists
DoesNotExist
Timeout
NoSuccess

Notes: - the default flag set is 'Read' (512).
- the semantics of the inherited constructors
apply

- DESTRUCTOR

Purpose: destroy the directory object

Format: DESTRUCTOR (in directory dir)

Inputs: dir: the object to destroy

Outputs: -

Throws: -

Notes: - the semantics of the inherited destructors apply

additional inspection emthods:

- get_size

Purpose: returns the number of bytes in the file

Format: get_size (in string name,
in int flags = None,
out int size);

Inputs: name: name of file to inspect

Outputs: size: number of bytes in the file

Throws: NotImplemented
IncorrectURL
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
DoesNotExist
Timeout
NoSuccess

Notes: - if 'name' can be parsed as URL, but contains an invalid entry name, an '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 flag set is 'None' (0).
- other flags are not allowed on this method, and cause an 'BadParameter' exception.
- similar to the 'st_size' field from 'stat' (2) as defined by POSIX

- is_file

Purpose: 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 string name,
in int flags = Read,
out directory dir)
Inputs: name: name of directory to open
flags: flags definition operation
modus
Outputs: dir: opened directory instance
PostCond: - the session of the returned 'dir' is that of
the calling directory instance.
Throws: NotImplemented
IncorrectURL
IncorrectSession
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
AlreadyExists
DoesNotExist
Timeout
NoSuccess
Notes: - all notes from the ns_directory::open_dir()
method apply.
- default flag set is 'Read' (512).

- open
Purpose: creates a new file instance
Format: open (in string name,
in int flags = Read,
out file file);
Inputs: name: file to be opened
flags: flags definition operation
modus
Outputs: file: opened file instance
PostCond: - the session of the returned 'file' is that of
the calling directory instance.
Throws: NotImplemented
IncorrectURL
IncorrectSession
AuthenticationFailed
AuthorizationFailed
PermissionDenied

```

BadParameter
IncorrectState
AlreadyExists
DoesNotExist
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 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 begin of the file (a seek(0,Start) is performed after the open).
 - the 'Binary' flag is to be silently ignored on systems which don't support it (i.e. non-Windows)>
 - the flag set 'Read | Write' is equivalent to the flag 'ReadWrite'.
 - default flag set is 'Read' (512).

```

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 Unix.

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, 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 session,
                          in string  url,
                          in int    flags = Read,
                          out file  obj)

```

```

Inputs: url:             location of file

```

```

        flags:          mode for opening
        session:        session to associate the
                        object with
Outputs: obj:          the newly created object
Throws: NotImplemented
        IncorrectURL
        AuthenticationFailed
        AuthorizationFailed
        PermissionDenied
        BadParameter
        IncorrectState
        AlreadyExists
        DoesNotExist
        Timeout
        NoSuccess
Notes:  - all notes from the directory::open() method
        apply.
        - the default flag set is 'Read' (512).

```

- DESTRUCTOR

```

Purpose: destroy the object
Format: DESTRUCTOR      (in file      obj)
Inputs: obj:            the object to destroy
Outputs: -
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:  -
Outputs: size:        number of bytes in the file
Throws:  NotImplemented
        IncorrectURL
        AuthenticationFailed
        AuthorizationFailed
        PermissionDenied
        BadParameter
        IncorrectState
        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 int len_in,
in array<byte> buffer,
out int len_out);

Inputs: len_in: number of bytes to be read

InOuts: buffer: buffer to read into

Outputs: len_out: number of bytes successfully
read

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
Timeout
NoSuccess

Notes: - 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, eg. at the end of the file.
- errors are indicated by returning negative
values for len_out, which correspond to
negatives of the respective 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, otherwise the
behaviour is undefined.
- if the file was opened in write-only mode (i.e.
no 'Read' or 'ReadWrite' flag was given, this
method throws an 'IncorrectState' exception.
- if len_in is smaller than 0, 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 int len_in,
in array<byte> buffer,
out int len_out);

Inputs: len_in: number of bytes to write
buffer: data to write

Outputs: len_out: number of bytes successfully written

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
Timeout
NoSuccess

Notes: - errors are indicated by returning negative values for len_out, which correspond to negatives of the respective 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 'IncorrectState' exception.
- if len_in is smaller than 0, 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'

Outputs: position: position of pointer after seek

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
IncorrectState
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 ERRNO error code.
- the file pointer can be positioned after the end of the file w/o extending it.
- the given offset can be positive, negative, or zero.
- note that read at or behind EOF return 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<ivec> ivec);
InOuts: ivec: array of ivec structs
defining start (offset) and
length (length) of each
individual read, buffer
to read into, and integer
to store result into.

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState

```

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 is thrown if any of the
          individual reads detects a condition which
          would raise an exception for the normal
          read() method.
        - a 'BadParameter' exception is thrown if any of
          the given ivecs has a negative len_in or offset.
        - similar to readv (2) as specified by POSIX

- write_v
Purpose: gather/scatter write
Format: write_v          (inout array<ivec>  ivec);
InOuts: ivec:           array of ivec structs
                                defining start (offset) and
                                length (length) of each
                                individual write, and
                                buffers containing the data
                                to write

Throws: NotImplemented
        AuthenticationFailed
        AuthorizationFailed
        PermissionDenied
        IncorrectState
        BadParameter
        Timeout
        NoSuccess
Notes: - the behaviour of each individual write is as
        in the normal write method.
        - an exception is thrown if any of the
          individual writes detects a condition which
          would raise an exception for the normal write
          method.
        - a 'BadParameter' exception is thrown if any of
          the given ivecs has a negative len_in or offset.
        - 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
Outputs: size:         size required for I/O
                        operation with that pattern

Throws:  NotImplemented
        AuthenticationFailed
        AuthorizationFailed
        PermissionDenied
        BadParameter
        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 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,
                        out  int      len_out);

Inputs:  pattern:      pattern specification for
                        read operation
InOuts:  buffer:       buffer to store read bytes
                        into
Outputs: len_out:      number of successfully read
                        bytes

Throws:  NotImplemented
        AuthenticationFailed
        AuthorizationFailed
        PermissionDenied
        BadParameter
        IncorrectState
        AlreadyExists
        DoesNotExist
        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.

```
- write_p
Purpose: pattern based read
Format: read_p      (in  string      pattern,
                    in  array<byte> buffer,
                    out int        len_out);

Inputs:  pattern:   pattern specification for
                    read operation
         buffer:    buffer to store read bytes
                    into
Outputs: len_out:   number of bytes successfully
                    written

Throws:  NotImplemented
        AuthenticationFailed
        AuthorizationFailed
        PermissionDenied
        BadParameter
        IncorrectState
        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.
```

Extended I/O methods:

```
- modes_e
Purpose: list the extended modes available in this
        implementation, and/or on server side
Format:  modes_e      (out array<string> emodes);
Inputs:  -
Outputs: emodes:      list of modes available for
                    extended I/O

Throws:  NotImplemented
        AuthenticationFailed
        AuthorizationFailed
        PermissionDenied
        IncorrectState
        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.

- read_e

Purpose: extended read

Format: read_e (in string emode,
in string spec,
inout array<byte> buffer,
out int len_out);

Inputs: emode: extended mode to use
spec: specification of read
operation

InOuts: buffer: buffer to store read bytes
into

Outputs: len_out: number of successfully read
bytes

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
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'.

- write_e

Purpose: extended write

Format: write_e (in string emode,
in string spec,
in array<byte> buffer,
out int len_out);

Inputs: emode: extended mode to use
spec: specification of write

		operation
	buffer:	buffer to store read bytes into
Outputs:	len_out:	number of successfully read bytes
Throws:	NotImplemented AuthenticationFailed AuthorizationFailed PermissionDenied BadParameter IncorrectState Timeout NoSuccess	
Notes:	<ul style="list-style-type: none"> - 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'. 	

3.10.3 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 (const char* url)
3  {
4    try {
5      // get type and other infos
6      saga::file my_file (url);
7
8      off_t size = my_file.get_size ();
9
10     if ( size > 10 )
11     {
12       char  buffer[11];
13       long  bufflen;
14
15       my_file.read (10, buffer, &bufflen);
16
17       if ( bufflen == 10 )

```

```
18     {
19         printf ("head: '%s'\n", buffer);
20     }
21 }
22 }
23
24 // catch any possible error - see elsewhere for better
25 // examples of error handling in SAGA
26 catch ( const saga::exception & e )
27 {
28     std::cerr << "Oops! SAGA error: " + e.what () + std::endl;
29 }
30
31 return;
32 }
```

3.11 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::attribute` interface. It is important to realize that this is intendet 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 probablility, 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 name space package. The interfaces presented here supplement the name space package with operations for operating on entries in replica catalogues.

3.11.1 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.

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 are identical. Often, one of these replicas is deemed to be a master copies (often its 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 namespace of logical files. Several replica system implementations have the notion of *containers*, 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::files` 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::attribute` interface. The method accepts patterns for meta data matches (`meta_pattern`) and for file name matches (`name_pattern`) and returns a list of logical file names for which both patterns match. The `meta_pattern` are formatted as defined for `find_attribute()` of the `saga::attribute` 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 `name_space` objects.

3.11.2 Specification

```

package saga.logical_file
{
    enum flags
    {
        None           = 0, // same as in name_space::flags
        Overwrite      = 1, // same as in name_space::flags
        Recursive      = 2, // same as in name_space::flags
        Dereference    = 4, // same as in name_space::flags
        Create         = 8, // same as in name_space::flags
        Excl           = 16, // same as in name_space::flags
        Lock           = 32, // same as in name_space::flags
        CreateParents  = 64, // same as in name_space::flags
        //             = 128, reserved for Truncate
        //             = 256, reserved for Append
        Read           = 512,
        Write          = 1024,
        ReadWrite      = 2048,
        //             = 4096 reserved for Binary
    }

    class logical_directory : extends      saga::ns_directory
                            implements    saga::attribute
                                // from ns_directory saga::ns_entry
                                // from ns_entry     saga::object

```

```

        // from ns_entry      saga::async
        // from object        saga::error_handler
    {

        CONSTRUCTOR    (in session      session,
                       in string       url,
                       in int          flags = Read,
                       out logical_directory dir);
        DESTRUCTOR     (in logical_directory dir);

        // add for inspection
        is_file        (in string       name,
                       out boolean     test);

        // open methods
        open_dir       (in string       name,
                       in int          flags = Read,
                       out logical_directory dir);

        open           (in string       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> meta_pattern,
                       in int          flags = None,
                       out array<string> names );
    }

class logical_file : extends    saga::ns_entry
                    implements  saga::attribute
                    // from ns_entry  saga::object
                    // from ns_entry  saga::async
                    // from object    saga::error_handler
{
    CONSTRUCTOR    (in session      session,
                   in string       url,
                   in int          flags = Read,
                   out logical_file file);
    DESTRUCTOR     (in logical_file file);

    // manage the set of associated replicas

```

```

    add_location    (in string      name);
    remove_location (in string      name);
    update_location (in string      name_old,
                    in string      name_new);
    list_locations  (out array<string> names);

    // create a new physical replica
    replicate      (in string      name,
                  in int          flags = None);

    // Attributes (extensible):
  }
}

```

3.11.3 Details

```
class logical_directory:
-----
```

This class represents a container for logical files in a logical file name space. It allows traversal of the catalogs 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    session,
                        in string      url,
                        in int         flags = Read,
                        out logical_directory
                                obj)
```

Inputs: session: session to associate with
 the object

url: location of directory

flags: mode for opening

Outputs: obj: the newly created object

Throws: NotImplemented
 IncorrectURL

```

AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
DoesNotExist
Timeout
NoSuccess
Notes: - the semantics of the inherited constructors
        and of the logical_directory::open_dir()
        method apply.
        - the default flag set is 'Read' (512).

- DESTRUCTOR
Purpose: destroy the object
Format: DESTRUCTOR      (in logical_directory obj)
Inputs:  obj:           the object to destroy
Outputs: -
Notes: - the semantics of the inherited destructors
        apply

- is_file
Purpose: alias for is_entry of saga::ns_directory

- open_dir
Purpose: creates a new logical_directory instance
Format:  open_dir      (in string name,
                       in int flags = Read,
                       out logical_directory dir);
Inputs:  name:         name of directory to open
         flags:        flags definition operation
                       modus
Outputs: dir:          opened directory instance
PostCond: - the session of the returned 'dir' is that of
           the calling logical_directory instance.
Throws:  NotImplemented
         IncorrectURL
         IncorrectSession
         AuthenticationFailed
         AuthorizationFailed
         PermissionDenied
         BadParameter
         IncorrectState
         AlreadyExists

```

DoesNotExist
Timeout
NoSuccess

Notes: - all notes from the ns_directory::open_dir()
method apply.
- default flag set is 'Read' (512).

- open

Purpose: creates a new logical_file instance

Format: open (in string name,
in int flags = Read,
out logical_file file);

Inputs: name: file to be opened
flags: flags definition operation
modus

Outputs: file: opened file instance

PostCond: - the session of the returned 'file' is that of
the calling logical_directory instance.

Throws: NotImplemented
IncorrectURL
IncorrectSession
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
AlreadyExists
DoesNotExist
Timeout
NoSuccess

Notes: - all notes from the ns_directory::open() method
apply.
- the flag set 'Read | Write' is equivalent to
the flag 'ReadWrite'.
- default flag set is '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> meta_pattern,
in int flags = None,
out array<string> names);

Inputs: name_pattern: pattern for names of

```

                                entries to be found
                                pattern for meta data of
meta_pattern:                  entries to be found
                                flags defining the operation
                                modus
flags:                          array of names matching both
Outputs: names:                pattern
Throws: NotImplemented
                                AuthenticationFailed
                                AuthorizationFailed
                                PermissionDenied
                                BadParameter
                                IncorrectState
                                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::attribute interface and for
          the find() method in the saga::ns_directory
          class apply. On conflicts, the find()
          semantics supercedes the find_attributes
          semantic. Only entries matching which would
          match in both the attribute and the name space
          find are returned.
        - the default flag set is 'None' (0).

```

```
class logical_file:
-----
```

This class provides means to handle the contents of logical files. That contents consists of strings representing locations of physical files (replicas) associated with the logical file.

- CONSTRUCTOR

Purpose: create the object

```
Format: CONSTRUCTOR (in string url,
                    in int flags = Read,
                    in session session,
                    out logical_file obj)
```

```
Inputs: url: location of directory
        flags: mode for opening
        session: session to associate with
```

```

Outputs:  obj:          the object
           obj:          the newly created object
Throws:   NotImplemented
          IncorrectURL
          AuthenticationFailed
          AuthorizationFailed
          PermissionDenied
          BadParameter
          IncorrectState
          AlreadyExists
          DoesNotExist
          Timeout
          NoSuccess
Notes:    - the semantics of the inherited constructors
           apply and of the logical_directory::open ()
           method apply.
           - the default flag set is 'Read' (512).

```

- DESTRUCTOR

```

Purpose:  destroy the object
Format:   DESTRUCTOR      (in logical_file  obj)
Inputs:   obj:            the object to destroy
Outputs:  -
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 string name);
Inputs:   name:           location to add to set
Outputs:  -
Throws:   NotImplemented
          IncorrectURL
          AuthenticationFailed
          AuthorizationFailed
          PermissionDenied
          BadParameter
          IncorrectState
          Timeout
          NoSuccess
Notes:    - this methods adds a given replica location

```

- (url) 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 location. The implementation documentation MUST specify how valid replica location are constructed.
 - if 'name' can be parsed as URL, but contains an invalid entry name, an 'BadParameter' exception is thrown.
 - if the replica is already in the set, this method does nothing, and in particular does not raise an 'AlreadyExists' exception
-
- remove_location
 - Purpose: remove a replica locate from the replica set
 - Format: remove_location (in string name);
 - Inputs: name: replica to remove from set
 - Outputs: -
 - Throws: NotImplemented
IncorrectURL
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
DoesNotExist
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 location. The implementation documentation MUST specify how valid replica location are constructed.
 - if 'name' can be parsed as URL, but contains an invalid entry name, an '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 that operation, the logical file object is still a valid object (see replicate() method description).
-
- update_location
 - Purpose: change a replica location in replica set
 - Format: update_location (in string name_old, in string name_new);
 - Inputs: name_old replica to be updated
name_new update for replica
 - Outputs: -
 - Throws: NotImplemented
IncorrectURL
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
DoesNotExist
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 location. The implementation documentation MUST specify how valid replica location are constructed.
 - if 'name' can be parsed as URL, but contains an invalid entry name, an 'BadParameter' exception is thrown.
 - if the old replica location is not in the set of locations, an 'DoesNotExist' exception is thrown, and the new replica location is not added.

 - list_locations
 - Purpose: list the locations in the location set

Format: list_locations (out array<string> names);
Inputs: -
Outputs: names: array of locations in set
Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
IncorrectState
Timeout
NoSuccess
Notes: - this method returns an array of strings
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).

- 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 string name,
in int flags = None);
Inputs: name: location to replicate to
flags: flags defining the operation
modus
Outputs: -
Throws: NotImplemented
IncorrectURL
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
AlreadyExists
DoesNotExist
Timeout
NoSuccess
Notes: - the method implies a two step operation:
1) copy any of the already associated replicas
to 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::local_file::add_location` methods apply.
 - the method is not required to be atomic, but: the implementation MUST be either successful 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 a descriptive error message.
 - the default flag set is `'None'` (0). The interpretation of flags is as described for the `ns_entry::copy()` method.
-

3.11.4 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.net/tmp/file.rep");
7
8     saga::file f ("gsiftp://localhost.net/tmp/file.rep");
9     std::cout << "size of local replica: "
10                << f.get_size ()
11                << std::endl;
12 }
```

3.12 SAGA Streams

A number of use cases involved 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 authorization and encryption schemes. The stream API is:

1. 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 performance implementations.
2. 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. 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 graduate 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.

3.12.1 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 implementation **MUST** have the same semantics on API level, i.e. behave like a reliable byte oriented data stream.

3.12.2 Stream States

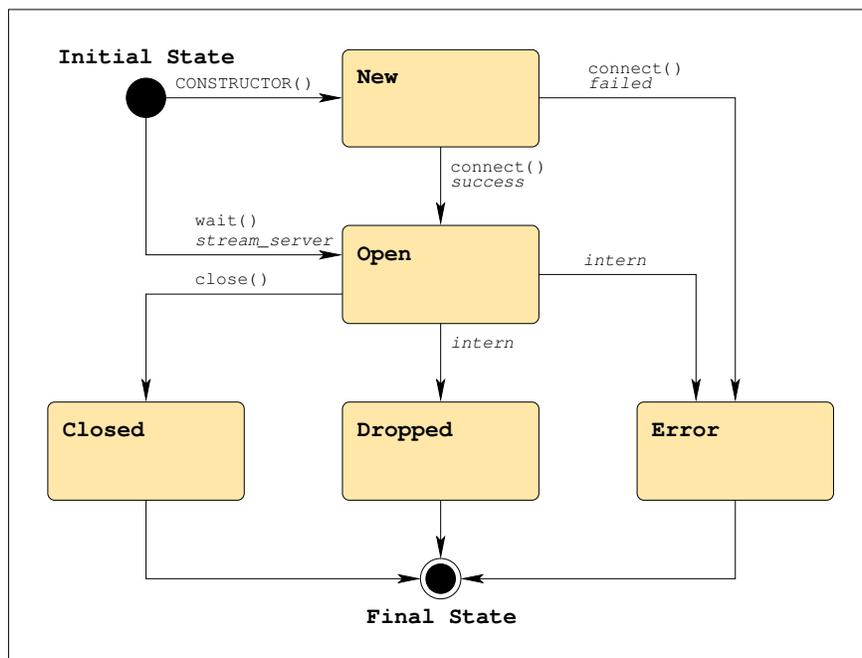


Figure 6: The SAGA stream state model (See figure 1 for a legend).

A SAGA stream can be in several states – the complete state diagram is shown in figure 6. The stream states are:

New: 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 occurs 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 occurred 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.

3.12.3 Stream Activity Types

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.

3.12.4 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
```

```

        // from object  saga::error_handler
    {
        CONSTRUCTOR      (in   session      session,
                        in   string        url,
                        out  stream_service obj);
        DESTRUCTOR      (in   stream_service obj);

        get_url         (out  string        url);

        serve           (in   float        timeout = -1.0,
                        out  stream        stream);

        close           (in   float        timeout = 0.0);

        // Metrics:
        //  name:  ClientConnect
        //  desc:  fires if a client connects
        //  mode:  Read
        //  unit:  1
        //  type:  Trigger
        //  value:
    }

class stream : extends    saga::object
               implements saga::async
               implements saga::attribute
               implements saga::monitorable
               // from object  saga::error_handler
    {
        // constructor / destructor
        CONSTRUCTOR (in   session      session,
                    in   string        url = "",
                    out  stream        obj);
        DESTRUCTOR  (in   stream        obj);

        // inspection methods
        get_url     (out  string        url);
        get_context (out  context       ctx);

        // management methods
        connect     (out  context       ctx);
        wait        (in   activity      what,
                    in   float         timeout = -1.0,
                    out  array<activity> activity);
        close       (in   float         timeout = 0.0);
    }

```

```
// I/O methods
read      (in   int           len_in,
           inout array<byte>  buffer,
           out  int           len_in);
write     (in   int           len_out,
           in   array<byte>   buffer,
           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 that 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
//        not 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 dependend
// notes: - the implementation MUST document the
//         default values for the available schemas
//
// name: Nodelay
// desc: determines if packets are sent
//       immediatley, i.e. w/o 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: State
// desc: fires if the state of the stream changes,
//       and has the value of the new state
//       enum
// mode: Read
// unit: 1
// type: Enum
// value: 'New'
//
// name: Read
// desc: fires if a stream gets readable
// mode: Read
// unit: 1
// type: Trigger
// value:
// notes: - a stream is considered readable if a
```

```
//          subsequent read() can successfully read
//          1 or more byte of data.
//
// name: Write
// desc: fires if a stream gets writable
// mode: Read
// unit: 1
// type: Trigger
// value:
// notes: - a stream is considered writable if a
//         subsequent write() can successfully write
//         1 or more byte of data.
//
// name: Exception
// desc: fires if a stream has an error condition
// mode: Read
// unit: 1
// type: Trigger
// value:
// notes: -
//
// name: Dropped
// desc: fires if the stream gets dropped by the
//       remote party
// mode: Read
// unit: 1
// type: Trigger
// value:
}
}
```

3.12.5 Details

```
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 session,
                               in string url = "",
                               out stream_service obj);
Inputs:  session:            session to be used for
                               object creation
          url:                channel name or url,
                               defines the source side
                               binding for the stream
Outputs: obj:                new stream_service object
PostCond: - the stream_service can now wait for incoming
           connections.
Throws:  NotImplemented
          IncorrectURL
          AuthenticationFailed
          AuthorizationFailed
          PermissionDenied
          Timeout
          NoSuccess
Notes:   - if the resource information given in the URL
           cannot ever be used by the implementation
           (e.g. hostname is not well formatted, scheme
           is not available), an 'IncorrectURL' exception
           is thrown, which must contain a detailed error
           message.
          - if the given url is an empty string (the
           default), the implementation will choose an
           appropriate default value.

```

- DESTRUCTOR

```

Purpose: Destructor for stream_service object.
Format:  DESTRUCTOR          (in stream_service obj)
Inputs:  stream:            object to be destroyed
Outputs: -
PostCond: - the stream is closed.
Throws:  -
Notes:   -

```

// inspection

```

- get_url
Purpose: get URL to be used to connect to this server
Format:  get_url            (out string url);
Inputs:  -
Outputs: url:                string containing the URL

```

of the connection.

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
IncorrectState
Timeout
NoSuccess

Throws: -

Notes: - returns a URL which can be passed to
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 client

Outputs: client: new Connected stream object

PostCond: - the returned client is in 'Open' state
- the session of the returned client is that of
the stream_server.

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
IncorrectState
BadParameter
NoSuccess

Notes: - if successful, it returns a new stream object
that is connected to the client.
- returns NULL or equivalent if it times out.
- 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),
an 'BadParameter' exception is thrown, which
must contain a detailed error message.
- in the asynchronous case, the resulting client
stream is passed as reference. That reference

```

        must be in the 'New' state - otherwise and
        'IncorrectState' exception is thrown.
    - if close() has been called on the
      stream_service before, an 'IncorrectState'
      exception is thrown.
    - for timeout semantics, see Introduction

- close
  Purpose:  closes an stream service
  Format:   close                (in float timeout)
  Inputs:   timeout              seconds to wait
  Outputs:  -
  PreCond:  - stream_service is serving
  PostCond: - no clients can be accepted anymore
  Throws:   NotImplemented
            IncorrectState
            NoSuccess
  Throws:   IncorrectState
  Notes:    - if a stream_service was closed earlier
            a 'IncorrectState' exception is thrown.
            - 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
              Introduction.
            - for timeout semantics, see Introduction.

```

```
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 an server.
  Format:   CONSTRUCTOR                (in session session,
                                       in string url,
                                       out stream stream);
  Inputs:   session:                  saga session handle

```

```

        url:                server location as URL
Outputs:  stream:          new, unconnected stream
                               instance
PostCond: - the state of the new socket is 'New'
Throws:   NotImplemented
          IncorrectURL
          AuthenticationFailed
          AuthorizationFailed
          PermissionDenied
          Timeout
          NoSuccess
Notes:    - server location and possibly protocol is
           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. hostname is not well formatted, 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 such 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 socket is only connected after the
             connect() method is called.

- DESTRUCTOR
Purpose:  destroy an stream object
Format:   DESTRUCTOR          (in stream obj)
Inputs:   obj:                stream to destroy
Outputs:  -
Throws:   -
Notes:    -

Inspection methods:
-----

- get_url
Purpose:  get URL used for creating the string
Format:   get_url            (out string url);
Inputs:   -
Outputs:  url:               string containing the URL

```

of the connection.

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
IncorrectState
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: -

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

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
IncorrectState
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 with as 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.
- 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: -
 - Outputs: -
 - PreCond: - the stream is in 'New' state.
 - PostCond: - the stream is in 'Open' state
 - Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
IncorrectState
BadParameter
Timeout
NoSuccess
 - Notes:
 - on failure, the stream state is changed to 'Error'
 - if the stream instance is not in 'New' state, and '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), an '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
 - Outputs: -
 - PreCond: - stream is in 'Open' state
 - PostCond: - stream is in 'Closed' state
 - Throws: NotImplemented
IncorrectState

Throws: NoSuccess
 IncorrectState
 Notes: - if a stream was closed earlier (i.e. is in 'Closed' or 'Dropped' state), this method does nothing, and, in particular, does not throw an 'IncorrectState' exception.
 - if the stream is in 'New' or 'Error' state, a 'IncorrectState' exception is thrown.
 - 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 Introduction.
 - for timeout semantics, see Introduction.

Stream I/O methods:

- read

Purpose: Read a raw buffer from socket.

Format: read (in int len_in, inout string buffer, out int len_out);

Inputs: len_in: Maximum number of bytes that can be copied in to the buffer.

In/Out: buffer: Empty buffer passed in to get filled

Outputs: len_out: number of bytes read, if successful. (0 is also valid)

Throws: NotImplemented
 AuthenticationFailed
 AuthorizationFailed
 PermissionDenied
 BadParameter
 IncorrectState
 Timeout
 NoSuccess

PreCond: - stream is in 'Open' state

Notes: - if the stream is blocking, the call waits until data get available.
 - if the stream is non-blocking, the call returns immediately, even if no data are

- available -- that is not an error condition.
- it is not an error to read less than len_in bytes.
- on read errors, a negative value for len_out is returned, which is equal to the POSIX errno value describing the error.
- a negative value for len_in will raise a 'BadParameter' exception.
- if the stream is not in 'Open' state, a 'IncorrectState' exception is thrown.

- write

Purpose: Write a raw buffer to socket.

Format: write (in int len_in,
in string buffer,
out int len_out);

Inputs: len_in: number of bytes of data in the buffer
buffer: raw array containing data that will be sent out via socket

Outputs: len_out: bytes written if successful

PreCond: - stream is in 'Open' state

Throws: NotImplemented
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
IncorrectState
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.
- on write errors, a negative value for len_out is returned, which is equal to the POSIX errno value describing the error.
- a negative value for len_in will raise a 'BadParameter' exception.
- if the stream is not in 'Open' state, a 'IncorrectState' exception is thrown.

```

- 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
  Outputs: cause:       activity type causing the
                        call to return

  PreCond: - stream is in 'Open' state
  Throws:  NotImplemented
           AuthenticationFailed
           AuthorizationFailed
           PermissionDenied
           IncorrectState
           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 availability of the socket
             (eg. OR'ed 'Read', 'Write', or 'Exception')
           - for timeout semantics, see Introduction
           - if the stream is not in 'Open' state, a
             'IncorrectState' exception is thrown.

```

3.12.6 Examples

Code Example

```

1  Sample SSL/Secure Client:
2  -----
3
4  Opens a stream connection using native security: context is
5  passed in implicitly via a global SAGA context
6  (GSI or SSL security)
7
8  // C++/JAVA Style
9  int recvlen;
10  saga::stream s ("localhost:5000");
11

```

```

12     s.connect ();
13     s.write  ("Hello World!", 12);
14
15     // blocking read, read up to 128 bytes
16     recvlen = s.read (buffer, 128);
17
18
19     /* C Style */
20     int recvlen;
21
22     SAGA_stream = SAGA_Stream_open ("localhost:5000");
23
24     SAGA_Stream_connect (s);
25     SAGA_Stream_write  (s, "Hello World!", 12);
26
27     /* blocking read, read up to 128 bytes */
28     recvlen = SAGA_Stream_read (s, buffer, 128);
29
30
31     c Fortran Style */
32     INTEGER  err,SAGAStrRead,SAGAStrWrite,err
33     INTEGER*8 SAGAStrOpen,streamhandle
34     CHARACTER buffer(128)
35     SAGAStrOpen("localhost:5000",streamhandle)
36     call SAGAStrConnect(streamhandle)
37     err = SAGAStrWrite(streamhandle,"localhost:5000",12)
38     err = SAGAStrRead(streamhandle,buffer,128)
39
40
41     Sample Secure Server:
42     -----
43
44     Once a connection is made, the server can use information
45     about the authenticated client to make an authorization
46     decision
47
48     // C++/JAVA Style
49     saga::stream_service server ("tcp://localhost/5000");
50     saga::stream          client;
51     int                    done = 0;
52
53     // now wait for a connection (normally in a loop)
54     do {
55         string value;
56
57         // wait forever for connection
58         client = server.serve (&ctx);
59
60         // get remote security details
61         saga::context ctx = client.get_context ();

```

```
62
63     // check if context type is X509, and if DN is the
64     // authorized one
65     if ( ctx.type () == saga::context::X509 &&
66         ctx.attribute_equals ("DN", auth_dn) )
67     {
68         done = 1; // allowed
69     }
70     else
71     {
72         SAGA::stream_close (client); // not allowed
73     }
74     } while ( ! done );
75
76     // start activity on client socket...
77
78
79     Example for async stream server
80     -----
81
82     // c++ example
83     class my_cb : public saga::callback
84     {
85     privat:
86         saga::stream_service ss;
87         saga::stream         s;
88
89     public:
90
91         my_cb (saga::stream_service ss_,
92              saga::stream         s_)
93         {
94             ss = ss_;
95             s  = s_;
96         }
97
98         ~my_cb (void) { }
99
100        bool cb (saga::monitorable mt,
101                saga::metric      m,
102                saga::context      c)
103        {
104            s = ss.serve ();
105            return (false); // want to be called only once
106        }
107    }
108
109    int main ()
110    {
111        saga::stream_service ss;
```

```
112     saga::stream      s;
113     my_cb cb (ss, s);
114
115     ss.add_callback ("client_connect", cb);
116
117     while ( true )
118     {
119         if ( s.state != saga::stream::Open )
120         {
121             // no client, yet
122             sleep (1);
123         }
124         else
125         {
126             // handle open socket
127             s.write ("Hello Client\r\n", 14);
128             s.close ();
129
130             // restart listening
131             ss.add_callback ("client_connect", cb);
132         }
133     }
134
135     return (-1); // unreachable
136 }
```

3.13 SAGA Remote Procedure Call

GridRPC is one of the few high level APIs that have been specified by the GGF [14]. 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-and-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 [14], 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-and-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:

<code>gridrpc</code>	-	<code>scheme</code>	-	identifying a grid rpc operation
<code>server.net</code>	-	<code>server</code>	-	server host serving the rpc call
<code>1234</code>	-	<code>port</code>	-	contact point for the server
<code>my_function</code>	-	<code>name</code>	-	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 a NULL reference,

and is to be created (e.g., allocated) by the SAGA implementation upon arrival of 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.)

3.13.1 Specification

```
package saga.rpc
{
  enum io_mode
  {
    In    = 1,      // input  parameter
    Out   = 2,      // output parameter
    InOut = 3       // input and output parameter
  }

  struct parameter
  {
    long      size; // number of bytes in buffer
    array<byte> buffer; // data
    io_mode   mode; // parameter mode
  }

  class rpc : implements  saga::object
              implements  saga::async
              // from object saga::error_handler
  {
    CONSTRUCTOR (in  session      session,
                 in  string       funcname = "",
                 out rpc          obj      );
    DESTRUCTOR  (in  rpc          obj      );

    // rpc method invocation
    call      (inout array<parameter> parameters );
  }
}
```

```
    // handle management
    close      (in   float           timeout = 0.0);
  }
}
```

3.13.2 Details

```
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: inits a remote function handle

Format: CONSTRUCTOR (in session session,
 in string funcname = "",
 out rpc obj);

Inputs: session: saga session to use
 funcname: name of remote method to
 initialize

Outputs: obj the newly created object

Throws: NotImplemented
 IncorrectURL
 AuthenticationFailed
 AuthorizationFailed
 PermissionDenied
 BadParameter
 DoesNotExist
 Timeout
 NoSuccess

Notes: - if the URL given as 'funcname' cannot ever be
 used by the implementation (e.g. hostname is
 not well formatted, scheme is not available),
 an 'IncorrectURL' exception is thrown, which
 must contain a detailed error message.

- if funcname is not given or an empty string (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 on 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
- Outputs: -
- PostCond: - the instance is closed.
- Throws: -
- Notes: - if the instance was not closed before, the destructor performs a close() of 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
- Outputs: -
- Throws: NotImplemented
IncorrectURL
AuthenticationFailed
AuthorizationFailed
PermissionDenied
BadParameter
DoesNotExist
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 param vector (like a non-zero size for a void buffer for an 'In' element), a 'BadParameter' exception is thrown.
 - arbitrary backend failures (e.g. semantic failures in the provided parameter stack, or any errors occurring during the execution of the remote procedure) MUST be mapped to a 'NoSuccess' exception, with an descriptive error message. That way, error semantics of the SAGA implementation and of the RPC function implementation are strictly distinguished.
- close
- Purpose: closes the rpc handle instance
- Format: close (in float timeout = 0.0);
- Inputs: timeout seconds to wait
- Outputs: -
- Throws: NotImplemented
IncorrectState
NoSuccess
- Notes:
- 'IncorrectState' is thrown if the object was closed before.
 - any subsequent method call on the object MUST also raise 'IncorrectState' (apart from DESTRUCTOR).
 - 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 Introduction.
 - for timeout semantics, see Introduction.
-

3.13.3 Examples

Code Example

```
1 // c++ example
2 // call a remote matrix multiplication A = A * B
3 try
4 {
5     rpc rpc ("gridrpc://fs0.das2.cs.vu.nl/matmul1");
6
7     std::vector <saga::rpc::parameter> params (2);
8
9     params[0].buffer = // ptr to matrix A
10    params[0].size   = sizeof (buffer);
11    params[0].mode   = saga::rpc::InOut;
12
13    params[1].buffer = // ptr to matrix B
14    params[1].size   = sizeof (buffer);
15    params[1].mode   = saga::rpc::In;
16
17    rpc.call (&params);
18
19    // A now contains the result
20 }
21 catch ( const saga::exception & e)
22 {
23     std::err << "SAGA error: " << e.what () << 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://fs0.das2.cs.vu.nl/matmul2");
33
34     std::vector <saga::rpc::parameter> params (3);
35
36     params[0].buffer = NULL; // buffer will be created
37     params[0].size   = 0;    // buffer will be created
38     params[0].mode   = saga::rpc::Out;
39
40     params[1].buffer = // ptr to matrix A
41     params[1].size   = sizeof (buffer);
42     params[1].mode   = saga::rpc::InOut;
43
44     params[2].buffer = // ptr to matrix B
45     params[2].size   = sizeof (buffer);
46     params[2].mode   = saga::rpc::In;
```

```
47     rpc.call (&params);
48
49     // params[0].buffer now contains the result
50 }
51 catch ( const saga::exception & e)
52 {
53     std::err << "SAGA error: " << e.what () << std::endl;
54 }
55
56 +-----+
57
58 // c++ example
59 // asynchronous version of A = A * B
60 try
61 {
62     rpc rpc ("gridrpc://fs0.das2.cs.vu.nl/matmul1");
63
64     std::vector <saga::rpc::parameter> params (2);
65
66     params[0].buffer = // ptr to matrix A
67     params[0].size   = sizeof (buffer);
68     params[0].mode   = saga::rpc::InOut;
69
70     params[1].buffer = // ptr to matrix B
71     params[1].size   = sizeof (buffer);
72     params[1].mode   = saga::rpc::In;
73
74     saga::task t = rpc.call <saga::task::ASync> (&params);
75
76     t.wait ();
77     // A now contains the result
78 }
79 catch ( const saga::exception & e)
80 {
81     std::err << "SAGA error: " << e.what() << std::endl;
82 }
83
84 +-----+
85
86 // c++ example
87 // parameter sweep example from
88 // http://ninf.apgrid.org/documents/ng4-manual/examples.html
89 //
90 // Monte Carlo computation of PI
91 //
92 try
93 {
94     std::string uri[NUM_HOSTS]; // initialize...
95     long times, count[NUM_HOSTS], sum;
```

```
97
98     std::vector <saga::rpc::rpc> servers;
99
100     // create the rpc handles for all URIs
101     for ( int i = 0; i < NUM_HOSTS; ++i )
102     {
103         servers.push_back (saga::rpc::rpc (uri[i]));
104     }
105
106     // create persistent storage for tasks and parameter structs
107     saga::task_container tc;
108     std::vector <std::vector <saga::rpc::parameter> > params;
109
110     // fill parameter structs and start async rpc calls
111     for ( int i = 0; i < NUM_HOSTS; ++i )
112     {
113         std::vector <saga::rpc::parameter> param (3);
114
115         param[0].buffer = i; // use as random seed
116         param[0].size   = sizeof (buffer);
117         param[0].mode   = saga::rpc::In;
118
119         param[1].buffer = times;
120         param[1].size   = sizeof (buffer);
121         param[1].mode   = saga::rpc::In;
122
123         param[2].buffer = count[i];
124         param[2].size   = sizeof (buffer);
125         param[2].mode   = saga::rpc::Out;
126
127         // start the async calls
128         saga::task t = servers[i].call <saga::task::ASync> (&param);
129
130         // save the task;
131         tc.add (t[i]);
132
133         // save the parameter structs
134         params.push_back (param);
135     }
136
137     // wait for all async calls to finish
138     tc.wait (-1, saga::task::All);
139
140     // compute and print pi
141     for ( int i = 0; i < NUM_HOSTS; ++i )
142     {
143         sum += count[i];
144     }
145
146     std::out << "PI = "
```

```
147         << 4.0 * ( sum / ((double) times * NUM_HOSTS))
148         << std::endl;
149     }
150     catch ( const saga::exception & e)
151     {
152         std::err << "SAGA error: " << e.what () << std::endl;
153     }
```

4 Intellectual Property Issues

4.1 Contributors

This document is the result of the joint efforts of many contributors. The authors listed here and on the title page are those committed to taking permanent stewardship for this document. They can be contacted in the future for inquiries about this document.

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

Thilo Kielmann

kielmann@cs.vu.nl
Vrije Universiteit
Dept. of Computer Science
De Boelelaan 1083
1081HV Amsterdam
The Netherlands

Andre Merzky

andre@merzky.net
Vrije Universiteit
Dept. of Computer Science
De Boelelaan 1083
1081HV Amsterdam
The Netherlands

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 acknowl-

edge, in alphabetical order, the contributions of:

Gabriele Allen (LSU), Stephan Hirmer (LSU), Hartmut Kaiser (LSU), Pascal Kleijer (NEC), 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).

4.2 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.

4.3 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.

4.4 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_SAGA` API.

Code Example

```
1
2 Example 1 (C++): Object State:
3 =====
4
5 // This example illustrates the expected life
6 // times of object states. State is shared in
7 // these cases, as only shallow copies occur.
8
9 int main (void)
10 {
11     { // task scope
12         saga::task t;
13
14         { // file scope
15             saga::file f;
16
17             { // session scope
18                 saga::session s;
19
20                 { // context scope
21                     saga::context c (saga::context::UserPass);
22
23                     s.add_context (c);
24                     f (s, "file:///tmp/data.bin");
25                     t = f.copy <saga::task::Task>
26                         ("file:///tmp/data.bak");
27
28                 } // leave context scope
29                 // session keep context state
30
31             } // leave session scope
32             // file keeps session state
33
34         } // file scope
```

```

35         // task keeps file state
36
37         t.run ();
38         // task runs, and uses state of file, of session,
39         // and of 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 char* url)
64 {
65     try {
66         saga::file my_file (url);
67
68         off_t size = my_file.get_size ();
69
70         if ( size > 10 )
71         {
72             char    buffer[11];
73             long    bufflen;
74
75             my_file.read (10, buffer, &bufflen);
76
77             if ( bufflen == 10 )
78             {
79                 std::cout << "head: " << buffer << std::endl;
80             }
81         }
82         else
83         {
84             std::cout << "head: file "    << file

```

```
85         << " is too short: " << size
86         << std::endl;
87     }
88 }
89
90 // catch any possible error - see elsewhere for better
91 // examples of error handling in SAGA
92 catch ( const saga::exception & e )
93 {
94     std::cerr << "Oops! SAGA error: " + e.what () + std::endl;
95 }
96
97 return;
98 }
99 -----
100 -----
101 Example 2b: C
102 -----
103 char* head (const char* url)
104 {
105     SAGA_File my_file = SAGA_File_create (url);
106
107     if ( NULL == my_file )
108     {
109         fprintf (stderr, "Could not create SAGA_File "
110                 "for %s: %s\n",
111                 url, SAGA_Session_get_error (theSession));
112         return (NULL);
113     }
114
115     off_t size = SAGA_File_get_size (my_file);
116
117     if ( size < 0 )
118     {
119         fprintf (stderr, "Could not determine file size "
120                 "for %s: %s\n",
121                 url, SAGA_Session_get_error (theSession));
122         return (NULL);
123     }
124     else if ( size > 10 )
125     {
126         char  buffer[11];
127         size_t buflen;
128
129         ssize_t ret = SAGA_File_read (my_file, 10, buffer,
130                                     &buflen);
131
132         if ( ret < 0 )
133         {
134             fprintf (stderr, "Could not read file %s: %s\n",
```

```
135         url, SAGA_Session_get_error (theSession));
136     return (NULL);
137 }
138
139     if ( buflen == 10 )
140     {
141         buffer [11] = '\0';
142         printf ("head: '%s'\n", buffer);
143         return (buffer);
144     }
145     else
146     {
147         fprintf (stderr, "head: short read: %d\n", buflen);
148         return (NULL);
149     }
150 }
151
152     fprintf (stderr, "head: file %s is too short: %d\n",
153             file, size);
154
155     return (NULL);
156 }
```

Example 2c: Java

```
161
162     import saga*;
163
164     class MyClass
165     {
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         string head (URI uri)
169         {
170             try
171             {
172                 saga::file f (uri);
173
174                 if ( 10 <= f.get_size () )
175                 {
176                     FileInputStream in (uri);
177                     byte[]          buffer = new buffer[10];
178                     int              res   = in.read (buffer);
179
180                     if ( 10 == res )
181                     {
182                         System.out.println ("head: " + buffer);
183                     }
184                     else
```

```
185     {
186         System.err.println ("head: read is short! " + res);
187     }
188
189     return new string (buffer);
190 }
191 else
192 {
193     System.out.println ("file is too small: " + size);
194 }
195 }
196
197 // catch any possible error - see elsewhere for better
198 // examples of error handling in SAGA
199 catch (...)
200 {
201     System.out.println ("Oops!");
202 }
203
204 return null;
205 }
206 }
```

Example 2d: Perl ('normal' error handling)

```
212
213 sub head ($)
214 {
215     my $url      = shift;
216     my $my_file = new saga::file (url)
217         or die ("can't create file for $url: $!\n");
218
219     my $size     = my_file->get_size ();
220
221     if ( size > 10 )
222     {
223         my $buffer = my_file->read (10)
224             or die ("can't read from file $url: $!\n");
225
226         if ( length ($buffer) == 10 ) )
227         {
228             print "head: '$buffer'\n";
229             return ($buffer);
230         }
231         else
232         {
233             printf "head: short read: %d\n" ($buffer);
234         }
235     }
236 }
```

```
235     }
236     else
237     {
238         print "file $url is too short: $size\n";
239     }
240
241     return (undef);
242 }
243
244 -----
245 Example 2e: Perl (exceptions)
246 -----
247
248 sub head ($$)
249 {
250     my $session = shift;
251     my $url      = shift;
252
253     eval
254     {
255         my $my_file = new saga::file (session, url);
256         my $size    = my_file->get_size ();
257
258         if ( size > 10 )
259         {
260             my $buffer = my_file->read (10);
261             my $buflen = length ($buffer);
262
263             if ( buflen == 10 )
264             {
265                 print "head: '$buffer'\n";
266                 return ($buffer);
267             }
268             else
269             {
270                 printf "head: short read: %d \n", length ($buffer);
271             }
272         }
273         else
274         {
275             print "file $url is too short: $size\n";
276         }
277     }
278
279     if ( $@ =~ /^saga/i )
280     {
281         print "caught saga error: $@\n" if $@;
282     }
283
284     return (undef);
```

```
285     }
286
287     -----
288     Example 2f: Fortran 90
289     -----
290
291     C Fortran 90 example
292     SUBROUTINE HEAD(session, url, buffer)
293
294     INTEGER      :: session, url, file, size, bufflen
295     CHARACTER*10 :: buffer
296
297     CALL SAGA_FILE_CREATE(session, url, file)
298     CALL SAGA_FILE_GET_SIZE(file, size)
299
300     IF size .GT. 10 THEN
301
302     CALL SAGA_FILE_READ(file, 10, buffer, bufflen)
303
304     IF bufflen .EQ. 10 THEN
305     WRITE(5, *) 'head: ', buffer
306     ELSE
307     WRITE(5, *) 'head: short read: ', bufflen
308     ENDIF
309     ELSE
310     WRITE(5, *) 'file is too short'
311     ENDIF
312
313     END
314
315     -----
316     Example 2g: Python
317     -----
318     # Python example
319     def head (session,url):
320
321     try:
322     my_file = saga.file(session,url)
323     size = my_file.get_size()
324
325     if (size > 10):
326     (buffer, bufflen) = my_file.read (10)
327     if (bufflen == 10):
328     print "head: ", buffer
329     return(buffer)
330     else
331     print "head: short read: ", bufflen
332     else
333     print "head: file is too short: ", size
334
```

```
335     # catch any possible error - see elsewhere for better
336     # examples of error handling in SAGA
337     except saga.Exception, e:
338         print "Oops! SAGA error: ", e.what()
339
340     +-----+
341
```

References

- [1] 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.
- [2] 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.
- [3] Babel Project. Scientific Interface Definition Language (SIDL). <http://www.llnl.gov/CASC/components/babel.html> .
- [4] 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/>.
- [5] DRMAA Working Group. Open Grid Forum. <http://forge.ogf.org/sf/projects/drmaa-wg/>.
- [6] 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.
- [7] Grid Checkpoint and Recovery Working Group (GridCPR), Open Grid Forum (OGF). <http://forge.ogf.org/sf/projects/gridcpr-wg>.
- [8] 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>.
- [9] 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.
- [10] JSDL Working Group. Open Grid Forum. <http://forge.ogf.org/sf/projects/jsdl-wg/>.
- [11] 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/> .

- [12] 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.
- [13] A. Merzky and S. Jha. A Requirements Analysis for a Simple API for Grid Applications. Grid Forum Document GFD.71, 2006. Global Grid Forum.
- [14] 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.
- [15] 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>.
- [16] 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.