Group: GGF SAGA-CORE-WG
Category: Recommendation
Title: SAGA API 1.0
Authors: Tom Goodale <goodale@cct.lsu.edu>
Shantenu Jha <s.jha@ucl.ac.uk>
Thilo Kielmann <kielmann@cs.vu.nl> ?????
Andre Merzky <andre@merzky.net>
John Shalf <jshalf@ncsa.uiuc.edu>
Christopher Smith <csmith@platform.com>
Date: May 01 2006
$Revision: 1.5 $

+--+

TODO: check with new GFD.63

Intellectual Property Statement
==========

+--+

Copyright Notice
==========

Contents

1 Introduction

This document describes SAGA version 1.0, 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 twofold:

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. (compared to serving 100 % of all possible requirements)
2. Provide a standardized, common interfaces across various grid middleware systems and their versions.

1.1 API Scope

It had been decided to orient SAGA's functionality on actual application needs. For this purpose, the SAGA group has collected an as broad as possible set of use cases. The received use cases have been published as GFD.xx [**?**]. From these use cases, the requirements on a SAGA API have been derived. The requirements analysis has been published as GFD.yy [**?**].

In detail, the SAGA-RG received about 12-15 use cases (with some overlap) in the latter half of 2004, in response to an open call for use cases. A list of the Use Cases can be found on the Wiki and the mailing list archives:

Wiki: http://wiki.cct.lsu.edu/saga/space/Use+Cases
email archive: http://www.gridforum.org/mail_archive/saga-rg/threads.html

In addition, several use cases were received by direct communication with other GGF groups: nine use cases origine from the GridRPC-WG. They have varying degrees of detail and completeness, and are available as *gridrpc_use_cases.zip* at the Wiki.

We have also looked at the OGSA use cases, published as GFD.29 [**?**]. The most significant OGSA use cases in GFD.29 were determined to be the two scientific grid use cases; Severe Storm Modeling and National Fusion Collaboratory. The OGSA use case document describes each use case at a rather high level, and unfortunately does not include API information.

There are also several groups in GGF that have worked on high level interfaces and API's; relevant API's and frameworks that have helped guiding the scope and designing the SAGA API can be found at: *http://wiki.cct.lsu.edu/saga/space/Related+Grid+APIs*.

In addition to these groups within the GGF, there are several projects and groups that have worked on APIs and frameworks that are similar in spirit to a SAGA API. These in turn have helped motivate a SAGA API. See the latter half of *http://wiki.cct.lsu.edu/saga/space/Related+Grid+APIs* for a partial listing. The work of all these groups has been considered while defining the SAGA API, and close collaboration with these groups ensures good mapping consistency between the APIs. Simply put, SAGA covers all those efforts, simplifies them, and provides a consistent look and feel. The SAGA Requirements Document [**?**] discusses this topic in more detail.

Based upon the use cases received and the discussions of the SAGA design team, and the group as a whole, it was felt that a two phase approach should be taken for developing the API.

1. The packages for phase (or version) one would be for the most significant and mature areas like file transfer, streams, job submission. They can be understood as those which are required the most, if not all use cases, which are well understood, and for which prototypes exist (at least partially). Along with this first version of areas, the API also needs to define necessary auxiliary APIs, such as as session and error handling etc.
   This document specifies exactly the API for these areas, being referred to as SAGA Version 1.0.
2. The packages for the second phase will be those which are to be found in some use cases only, like computational steering, or for an advert service.

Based upon the above reasoning, areas of functionality that were included in SAGA Version 1.0 are the following. They are also referred to as SAGA's *packages*.

- jobs
- files (and logical files)
- streams
- auxiliary API's for
  - session handle and security context
  - errors
  - asynchronous method calls (tasks)
  - attributes

Possible areas of functionality (packages) to be included in future SAGA Versions are:

- steering and monitoring
- possibly combining logical/physical files (read on logical files)
- advert service (see GAT [**?**]: persistent information service)
- GridRPC [**?**]
- GridCPR [**?**]
- Task dependencies (simple work flows and batches)
- extensions to existing classes

The versions as described 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.

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).

```
Style and Design Issues:
------------------------

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, bothe for
simplicity and also to avoid complications when mapping to a
procedural language.

The design team chose SIDL (Scientific Interface Definition
Language; http://www.llnl.gov/CASC/components/babel.html) to
specify the API.  This provides a language neutral
represention of the API, but with well-defined syntax and
clear mapping to implementation languages.

The need for asynchroneous calls was explicitely stated by the
user community, as reasonable synchroneous behaviour cannot
always be expected in Grids.  The team discussed the merits of
```

callback and polling mechanism and agreed that a non-blocking polling mechanism would be used initially. The SAGA task interface provides a mechanism to associate an asynchronous call with each blocking API call, and a polling mechanism to determine the state of the task; in the future a callback mechanism may augment this.

Object Oriented versus Procedural, Language issues:

The abstract SAGA API specification is object oriented, and specified in SIDL. Normative bindings for specific languages, both object oriented and procedural, will be defined in additional documents.

In several places, 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.

The document contains several examples illustrating the use of the APIs, and these have naturally been shown using specific languages, such as C++. These examples should not be taken as normative, but merely as illustrative of the use of the API. When normaltivee language bindings are available these examples may be revised to reflect these bindings. In order to give a feeling 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.

SIDL

As stated above, SIDL was choosen as specification language for this API specification. However, the document stays not true to the SIDL language in several places. This section gives a very short introdution to SIDL, and also lists a number of 'Notes to Implementors' on how to interprete 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: a set of methods
 - class:     stateful object and set of methods
 - method:    a service a caller can invoke on a object
 - type:      contraint to value of method parameters

SIDL supports single inheritance of classes, and multiple
inheritance of interfaces.  Method definitions have
signatures, which define what parameters are accepted on
method invocation.  These parameters can be

 - in:    passed by value
 - out:   passed by reference, assumed NULL
 - inout: passed by reference

A implementation can destroy and re-allocate parameters which
are passed by reference, no assumptions in that respect should
be made to the implementation.


Notes to implementors:
----------------------

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 does also have the notion of 'versions', which are
actually required on packages.  We do not use versions in this
specification, as the specification itself is versioned.

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 native types.  a char* is expressed in this document
as array<byte,1> .

This specification defines all method calls as VOID (or rather
does not specify a return type for any method call at all).
Instead we define out parameters.  Out parameters in SIDL are
passed by reference.  However, for this specification we
expect language bindings to use one output parameter as return
value to function calls where appropriate.

We are using output parameter for the following reason: most
calls in the SAGA specification can be use asynchroneously.
As such, they return a task object, and have additional output
```

parameters as described in the spec. Only the synchroneous versions do not return tasks, and can hence use normal output parameters.

+---+

Compliant Implementations:
==========

A implementation of the SAGA API is "SAGA compliant" if it implements all objects and methods of the SAGA API specification. However, the implementation MAY throw a NOT_IMPLEMENTED exception for calls; however, that exception is to be used only in necessary cases, for example if a underlying Grid Middleware does not provide some capability, and if that capability can also not be emulated. The implementation MUST carefully document and motivate the use of the NOT_IMPLEMENTED exception.

The semantics for all methods is explicitely described for all methods, and must be followed by compliant implmentations, unless explicitely stated otherwise.

Also, a compliant implementation MUST follow the SAGA API specification both syntactically and semantically. The consistency model supported by the implementation MUST be documented. The thread safety or unsafety of the implementation MUST be documented. The underlying middleware security model MUST be documented, as MUST its mapping to the SAGA::Context object.

A implementation is "partial SAGA compliant" if it follows the comformance guidlines above, but implements only some packages (some objects are not implemented).

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

Please note that the current specification does not as of yet define 'subsystens' sufficiently well -- that will be fixed in the final specification.

Notes to implementors:

Early versus late bindings:

An implementation may choose to use late binding to middleware. That 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 that file might be performed with GridFTP.

  Late bindings 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 file write might fail
  on HTTP for some reasons, but might succeed via GridFTP.  The
  situation might be inversed for file reads.  In order to allow
  alternating access via both protocols, the file pointer
  information (e.g. the file object state) MUST be hold on
  client side.

  It is left to the later experience documents about the SAGA
  API implementations to discuss potential problems arising from
  early/late binding implementations.


+-------------------------------------------------------------+


Security considerations:
========================

  As the API is supposed to get implemented on very 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
  SAGA.Context for details.  A SAGA implementation is considered
  secure if it fully supports/implements the security model of
  the middleware layer(s) it depends upon, and does not provide
  any (intentional and unintentional) means to bypass that
  security model, and does not weaken the security model
  policies.

+-------------------------------------------------------------+


Relation to OGSA:
=================

  Although we feel that widely off target, the SAGA API
  specification effort has often been compared to, and seen in
  rivalry to the OGSA standardization effort.  That is NOT
  correct.  Reasons are the following:

    - OGSA strives to define interfaces on Service and middleare
      level,

      SAGA aims on application level.
```

```
    - 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 is an Architecture (or a framework for an
      architecture).

      SAGA is an API.


    - OGSA cannot sensibly interface to SAGA.

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


    - The OGSA spec aims at middleware developers.

      The SAGA spec aims at application developers.

  For these and more reasons we think that SAGA and OGSA are
  complementary, but by NO means competetive.  The only
  communality we are aware of is the broadness of both
  approaches: both OGSA and SAGA strive to cover more than one
  specific area of, well, middleware and application,
  respectively.

+-----------------------------------------------------------------+


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 an 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 protocoll is,

  However, that innocent looking problem has farreaching
  consequences, and in fact is, to our best knowledge, unsolved.
  Consider the following server setup on host.net:
```

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

  The entity described by the two URLs

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

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

  Similar consideration hold for absolute file names, and for
  file names relative to the users home dir: consider

    httpd://host.net/\~{}merzky/tmp/file

  That URL point on my linux box to

    file:///home/merzky/public\_html/tmp/file

  and not, as could have been expected, to

    file:///home/merzky/tmp/file

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

  Further, even if a correct translation of protocols and hence
  URLs suceeds, there is no guarantee that the referred file is
  actually available via that protocol -- that 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.
     Futher, that 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).  An SAGA
     compliant implementation MAY be able to choose a suitable
     protocol.

  3) Abstract name spaces, such as the name space used by
     Replica Systems, or by Grid File Systems, efficiently and
     transparently hides that problem from the end user.  SAGA
```

```
     encourages implementations to use such name spaces.

  4) A URL which cannot be handled for the stated reasons MUST
     cause the exception Incorrect\_URL to be thrown.  Note that
     this holds only for those cases, where a given URL cannot
     be handled \emph{as such}, e.g. because the protocol is
     unsupported, any:// cannot be handled, or a necessary URL
     translation failed.  Any other error related to the URL
     (e.g. file at service is not available) MUST be indicated
     by the error codes/exceptions as state at the method
     specifications in this document.

+---------------------------------------------------------------+


Additional Notes
================

  - For files, flags are used to specify if a 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
    that seeming inconsistency.  However, we think that a
    forcefull unification of both schemes would be more awkward
    to use, and ate the same time less useful.

  - About consistency we had a lengthy discussion, with the
    aggreement that the consistency model is to be defined and
    documented by the implementation.  The API spec itself does
    not assume any specific consistency model, as we feel that
    (a) POSIX consistency is not achievable within reasonable
    effort/performance, (b) if the user assumes the worst (no
    consistency), he will still be able to make good use of the
    API, and (c) reality will be somewhere in the middle.


+---------------------------------------------------------------+
```

```
+--------------------------------------------------------------+

        #####
       #     #  #####        #  ######   ####   #####
       #     #  #    #       #  #       #    #    #
       #     #  #####        #  #####   #         #
       #     #  #    #       #  #       #         #
       #     #  #    #  #    #  #       #    #    #
        #####   #####    ####   ######   ####     #

+--------------------------------------------------------------+


Summary:
========

  The basic SAGA object interface provides methods which are
  essential for all SAGA objects.  For now, it provides a unique
  ID which helps to maintain list of SAGA objects in the
  application, and inspection to allow to test for the objects
  type and the attached session.

+--------------------------------------------------------------+


Specification:
==============

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

```
    interface object
    {
      get_id       (out int         id     );
      get_type     (out object_type type   );
      get_session  (out session     session);
    }
  }
```

```
+-------------------------------------------------------------+
```

```
Details:
========

  class object:

    - get_id:
      Purpose: query the object ID
      Format:  get_id               (out int id);
      Inputs:  none
      Outputs: id                    id of object

    - get_type:
      Purpose: query the object type
      Format:  get_type             (out object_type type);
      Inputs:  none
      Outputs: type                  type of object

    - get_session:
      Purpose: query the objects session
      Format:  get_session          (out session s);
      Inputs:  none
      Outputs: s                     session of object
      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.  The session
                 returned is NULL then (that includes the
                 session object itself).
```

```
+-------------------------------------------------------------+
```

```
Examples:
=========

  // c++ example

  // have 2 objects, streams and files, and do:
  //  - read 100 bytes
```

```
  //  - skip 100 bytes
  //  - read 100 bytes

  char out1[100];
  char out2[100];
  char out_last[100];

  // create map
  std::map <saga::task, saga::object>  tmap;

  // create objects, and map
  saga::file   f (url[1]);
  saga::stream s (url[2]);

  s.connect ();

  // create tasks for reading first 100 bytes, and map
  saga::task t1 = f.read <saga::task> (100, buf1, &out1);   tmap[t1] = f;
  saga::task t2 = s.read <saga::task> (100, buf2, &out2);   tmap[t2] = s;

  // put in same container...
  saga::task_container tc;

  tc.add (t1);
  tc.add (t2);

  // ... and wait who gets done first
  while ( saga::task t = tc.wait () )
  {
     // depending on type, skip 100 byte then create a
     // new task for the next read, and re-add to the tc

     if ( tmap[t].get_type () == saga::object::File )
     {
       saga::file (tmap[t]).seek (100, SEEK_SET);
       tc.add (saga::file (tmap[t]).read <saga::task> (100, &out_last))
     }
     else
     if ( tmap[t].get_type () == saga::object::Stream )
     {
       saga::stream (tmap[t]).read (100, NULL); // ignore result
       tc.add (saga::stream (tmap[t]).read <saga::task> (100, &out_last))
     }
     else
     {
       throw saga::exception ("Something went terribly wrong");
     }

     // tc is filled again, we run forever, read/seeking from
     // whoever we find after the wait.
  }
```

```
+-------------------------------------------------------------+


Notes:
======

  Really useful will the base saga object once we add
  serialization - it will then be possible to serialize objects
  (e.g. tasks!), and to resume operations on them at a later
  point.


+-------------------------------------------------------------+
```

```
-------------------------------------------------------------+

      #####
     #     #  ######   ####    ####   #   ####   #    #
     #        #       #       #       #  #    #  ##   #
      #####   #####    ####    ####   #  #    #  # #  #
           #  #            #       #  #  #    #  #  # #
     #     #  #       #    #  #    #  #  #    #  #   ##
      #####   ######   ####    ####   #   ####   #    #

-------------------------------------------------------------+
```

Summary:
========

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).

```
+-------------------------------------------------------------+
```

Specification:
==============

```
package saga.session
{
  class session
  {
    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);
  }
}
```

```
+-------------------------------------------------------------+
```

Details:
========

class session:

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

```
A context (which encapsulates security information in SAGA)
can be attached to a session.  A SAGA implementation MAY
allow to attach more than one context to a single session.
However, a single context instance can get attached only
once to a specific session instance.

A session can be used as first parameter to all SAGA object
instantiation calls.  SAGA objects created from other SAGA
objects inherit its session.  Only some objects do not need
a session handle on creation time, and can hence be shared
between sessions.  That includes:
 - context
 - job_description
 - metric
 - utility classes

If the session handle is omitted as first parameter, a
default session handle is used, with default security
context(s) attached. Example:

  // 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);

Any SAGA operation CAN throw a IncorrectSession exception if
involves two different session handles.


- CONSTRUCTOR
  Purpose: create the object
  Format:  CONSTRUCTOR          (out session obj)
  Inputs:  none
  Outputs: obj:                  the newly created object

- DESTRUCTOR
  Purpose: destroy the object
  Format:  DESTRUCTOR           (in  session obj)
  Inputs:  obj:                  the object to destroy
  Outputs: none

- add_context
  Purpose: attach a security context to a session handle
  Format:  add_context          (in context  context);
  Inputs:  context               Security context to add
  Outputs: none
  Throws:  BadParameter
           AlreadyExists
```

```
    - remove_context
      Purpose: detach a security context from a session handle
      Format:  add_context          (in context  context);
      Inputs:  context               Security context to remove
      Outputs: none
      Throws:  BadParameter
               DoesNotExist

    - list_contexts
      Purpose: retrieve all contexts attached to a session
      Format:  list_contexts        (out array<context>
                                             contexts);
      Inputs:  none
      Outputs: contexts              list of contexts of this
                                     session
      Note:    - a empty list is returned if no context is
                 attached, yet.
               - contexts may get added to a session by default.
                 hence the returned list MAY be non empty even
                 if no add_context was ever called before.

+-------------------------------------------------------------+


Examples:
=========

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

  s.add_context   (c);

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

  // file has same session attached as dir


+-------------------------------------------------------------+

Notes:
======

  Most libraries use session handles to distinguish scope
  (security, settings, lifetime) of objects etc.

  GAT used a context object, which is a session handle
  with attached information (security context, preferences)
  and some methods (get 'self', init environment, current
  state, ...).
```

```
  Proposal: re-use what we have in GAT: falls back to the well
  known paradigm of a session handle), plus give
  potential for a few more features (see above).

+-------------------------------------------------------------+
```

```
+-------------------------------------------------------------+

    #####
   #     #   ####   #    #   #####  ######  #    #   #####
   #         #    #  ##   #     #    #        #  #      #
   #         #    #  # #  #     #    #####     ##       #
   #         #    #  #  # #     #    #         ##       #
   #     #   #    #  #   ##     #    #        #  #      #
    #####    ####   #    #     #    ######  #    #     #

+-------------------------------------------------------------+
```

Summary:
========

The context 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 instanciated in that session. Multiple contexts can co-exist on one handle. A single context can be shared between threads and sessions. SAGA Objects created from other SAGA Objects inherit its session and also its context(s).

A implementation CAN implement various types of Security contexts. or just one type. The type of context to be created is specified by a enum which is the only argument to the context constructor. The default type is unknown. Other methods than get_type MUST NOT be called on a context with type Unknown.

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 set or requested.

For incoming interactions (streams, monitoring, steering), read only contexts are used to inform the application about the requestor idendity. To support that, a number of specific getter methods are available.

```
+-------------------------------------------------------------+
```

Specification:
==============

```
package saga.context
{
  enum context_type
  {
```

```
      Unknown         = -1,
      X509            =  1, // Globus
      MyProxy         =  2, // X509 extended
      KeyStore        =  3  // Unicore
      SSH             =  4, // SSH
      Kerberos        =  5, // Kerberos
      UserPass        =  6, // Unix default
    }

    class context : implements-all saga::object
                    implements-all saga::attribute
                    implements-all saga::monitorable
    {
      CONSTRUCTOR (in  context_type type,
                   in  session      session);
                   out context      context);
      DESTRUCTOR  (in  context      context);

      get_type    (out context_type type);
    }
  }
```

```
+-------------------------------------------------------------+
```

```
Details:
========

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

    - CONSTRUCTOR:
      Purpose: create a security context
      Format:  CONSTRUCTOR          (in  context_type type,
                                     in  session      session,
                                     out context      context);
      Inputs:  type                  type of context
      Outputs: context               the newly created context
      Throws:  BadParameter
      Notes:   - if NO session handle is defined for at context
                 creation, the resulting context is NOT bound
                 to any session, any must be added to a session
                 in order to get used.
               - 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: none
```

```
    - get_type:
      Purpose: query the context type
      Format:  get_type              (out context_type type);
      Inputs:  none
      Outputs: type                  type of context

+-------------------------------------------------------------+

Examples:
=========

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

  s.add_context (c);

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

  // file has same session attached as dir

+-------------------------------------------------------------+


Notes:
======

  - Following attributes MUST be supported by the
    correponding context types:

      Unknown:
        No attributes supported

      X509:
        Set/Get:
          X509_Proxy          (/tmp/x509...)
          X509_CertDir        (/etc/grid-security/certificates/)
        Get:
          X509_RemoteDN
          X509_RemoteHost
          X509_RemotePort

      MyProxy
        Set/Get:
          MyProxy_UserName    (anonymous)
          MyProxy_Password    (anon)
        Get:
          MyProxy_RemoteUserName
```

```
          MyProxy_RemoteHost
          MyProxy_RemotePort

      KeyStore
        Set/Get:
          KeyStore_Location    ($HOME/.keystore)
          KeyStore_Password    (anon)
        Get:
          KeyStore_RemoteUserName
          KeyStore_RemoteHost
          KeyStore_RemotePort

      SSH
        Set/Get:
          SSH_PrivKey          ($HOME/.ssh/id_dsa)
          SSH_PublKey          ($HOME/.ssh/id_dsa.pub)
        Get:
          SSH_RemoteUserName
          SSH_RemoteHost
          SSH_RemotePort

      Kerberos
        Set/Get:
          Kerberos_Ticket      (/tmp/kticket...) ?
        Get:
          Kerberos_RemoteUserName
          Kerberos_RemotePort
          Kerberos_RemotePort

      UserPass
        Set/Get:
          UserPass_UserName    (anonymous)
          UserPass_Password    (anon)
        Get:
          UserPass_RemoteUserName
          UserPass_RemoteHost
          UserPass_RemotePort

      - Other types MAY be specified by a SAGA
        implementation.

      - Default values can be specified by a SAGA
        implementation.

  - Should we also specify the default values?  Mostly
    simple I guess.  But then the defaults may  differ per
    platform and installation, so leaving that to the
    implementation gives more flexibility...

+-------------------------------------------------------------+
```

```
Examples:
=========

  // c++ example
  saga::context c_1 (saga::context::SSH);  // default attribs apply
  saga::context c_2 (saga::context::FTP);

  c_2.set_attribute ("UserName", "myself");
  c_2.set_attribute ("Password", "secret");

  saga::session s;
  s.add_context (c_1);
  s.add_context (c_2);

  saga::file f ("any://remote.net/tmp/data.txt", s);

  // file can be accessed now via ssh or ftp

+-------------------------------------------------------------+


Notes:
======

  For encapsulating security information, a security context
  is created and associated with a context (aka session
  handle).  The security context can hold information about
  X509 certificates, private/public keys, username/password,
  kerberos tickets etc., and provides these information to
  the SAGA implementation as needed.

  A SAGA implementation MAY be able to attach more than one
  security context to one context.

  Moved from Stream:
  ------------------

  We need to do something with a SAGA context and security
  contexts.

    context: This is an opaque datastructure that is used
             throughout the SAGA APIs.  It hides key state
             information such as the security context and
             other shared data.  It is passed in explicitly
             in order to support thread safety.


  interface security_info:

    security_info encapsulates information about the host or
```

```
    authenticated user on the other end of a stream/socket
    connection.

    The information encapsulated by this object can be used to
    make authorization/access-control decisions based on the
    identity of the remote user or host.

    The security_info is an opaque structure that can be
    interrogated (via a different api) to determine the identity
    of the connected host.  This information is essential for
    supporting Authorization and access control mechanisms.
    convenience functions that encode some of the most commonly
    required information used to make authorization decisions.
    Additional information that can be used to make authorization
    decisions or provide other identifying features for the
    remotely connected host or user can be interrogated using the
    SAGA "parameters" API that the security_info object
    implements.  These parameters are always interrogated as
    string-based Key-value pairs.

    - get_remote_user_name
      Purpose: Gets the username associated with the remotely
               connected entity (if available).
      Format:  get_remote_user_name    (out string username);
      Inputs:  none
      Outputs: username:             username assoc with remote
                                     connection
      Notes:   - returns NULL string if UserName not available.

    - get_remote_dn
      Purpose: Gets the distinguished name associated with the
               remotely connected entity (if available).
      Format:  get_remote_dn          (out string dn);
      Inputs:  none
      Outputs: dn                    Distinguished Name assoc
                                     with remote connection
      Notes:   - returns NULL string if that information is not
                 available.

    - get_remote_host
      Purpose: Gets the hostname of the other side of
               connected stream (if available).
      Format:  get_remote_host        (out string hostname);
      Inputs:  none
      Outputs: hostname:             hostname assoc with remote
                                     connection
      Notes:   - returns NULL string if that information is not
                 available.

    - get_remote_port
      Purpose: Gets the portnumber of the other side of
```

```
                connected stream (if available).
      Format:   get_remote_port        (out int port);
      Inputs:   none
      Outputs:  port:                  portnumber assoc with remote
                                       connection
      Notes:    - returns '0' if that information is not
                  available.

+-------------------------------------------------------------+
```

```
+-------------------------------------------------------------+

          #######
          #       #####   #####    ####   #####
          #       #    #  #    #  #    #  #    #
          #####   #    #  #    #  #    #  #    #
          #       #####   #####   #    #  #####
          #       #   #   #   #   #    #  #   #
          #######  #    #  #    #   ####   #    #

+-------------------------------------------------------------+

Summary:
========

  Each SAGA API call has an associated list of exceptions it can
  throw.  These exceptions implement the saga::exception
  interface.

  All objects in SAGA implement 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 will provide a
  mechanism to examine nested exceptions.

  The details of the error handling mechanisms will be defined
  in the respective language bindings.

+-------------------------------------------------------------+

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);
      get_error_message    (out string         message);
      has_error_type       (out boolean        state);
    }
  }

+-------------------------------------------------------------+

Details:
========

  SAGA provides a set of well defined error states (exceptions) which
  MUST be supported by the implementations.  As to wether these error
  states are non-critical, critical or fatal depends on (A) the
  specific implementation (one implementation might be able to recover
  from an error while another implementation might not), (B) the
  specific application use case (for example the error 'file does not
  exist' is fatal or not depends if the application really needs
  information from that file).

  The exceptions available in SAGA are listed below, with a
  number of explicit examples on when that exception should be
  thrown.  These examples are not normative, but illustrative.
  The spec defines the set of allowed exceptions for each method
  explictly - that set is normative.

  The exceptions below are sorted, most specific ones first,
  least specific ones last.  On any error condition, the most
  specific exception possible MUST be thrown.

  The SAGA spec defines what exceptions can be thrown by what
  method.  However, depending on the implementation, other
  exceptions can be thrown as well.  For example, an
  implementation might have authorization on specific attribute
  settings, and could throw a AuthorizationFailed exception on
  attempts to write that attribute, even if that is not specified
  in the SAGA spec.  New SAGA exception types SHOULD NOT be
  defined by the implementation.

  Depending on the language bindings, the listed exceptions are
  derived from the base SAGA exception types, or are error codes
  with that specific name etc.  For details, see the language
  bindings.


  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.

IncorrectSession:

A method was invoked which effects two object instances which belong to different SAGA sessions. Currently, the SAGA API does not provide any method which could potentially have colliding sessions - that exception is defined for future SAGA extensions, e.g. work flows.

AuthentificationFailed:

An operation failed because none of the available contexts of the used session could be used for successful Authenitification.

Example:
- a remote host did not accept a X509 certificate because the respective CA is unknown there.

AuthorizationFailed:

An operation failed because none of the available contexts of the used session could be used for successful Authorization. Authentification succeeded.

Example:
- although a certificate was valid on a remote GridFTP server, the ID could not be mapped to a valid local user ID.

PermissionDenied:

A operation failed because the identity used for the operation did not have sufficient permissions to perform the operation successfully. Authentification and authorization have been successful.

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.

```
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 | WriteOnly)


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, if for example they indicate that
  overwrite is allowed.

  Examples:
   - a file to be created already exists
   - a target for a file move 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
    - a context to be added to a session was added earlier


  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,
    if they for example 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 replicated is not in a replica set
     - 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 by id 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 a IncorrectState exception.

    Examples:
     - attempt to change an attribute which is read only
     - attempt to change or update a metric which is read only


  ReadError:
  ----------

    This exception indicates that a read operation on a file,
    directory or stream failed, although the object in
    question has been in the correct state (i.e.  readable).
    On NonBlocking objects, reads might frequently fail but
    might succeed in a later call (EAGAIN) - in such cases
    this exception MUST NOT be thrown, as that situation does
    not indicate an error.
```

```
    Examples:
     - a read on a file failed because the file was opened
       read-only
     - a read on a stream failed because no data are available


  WriteError:
  -----------

    This exception indicates that a write operation on a file,
    directory or stream failed, although the object in question
    has been in the correct state (i.e.  writable).  On
    NonBlocking objects, writes might frequently fail but might
    succeed in a later call (EAGAIN) - in such cases this
    exception MUST NOT be thrown, as that situation does not
    indicate an error.


  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 times out - that is indicated by
    the waits return value, and does not pose an error
    condition.  The time waited before a Timeout is indicated
    depends on the implementation and on the backend, and
    SHOULD be documented where it is implementation specific.

    Examples:
     - a remote file authorization request timed out
     - a remote data base access timed out
     - a host name resolution timed out
     - a started 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.

    Examples:
     - a once open file is not available right now
     - a backend did not answer a request about job state
     - a file copy was interrupted mid-stream
```

```
  class exception:
  ----------------

    This is the basic exception interface for all exceptions
    thrown by a SAGA object implement.

    Note that saga::excpetion 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


    - DESTRUCTOR
      Purpose: destroy the exception
      Format:  DESTRUCTOR    (in  exception e);
      Inputs:  e              the exception to destroy
      Outputs: none


    - 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:  none
      Outputs: message        the error message


    - get_object
      Purpose: gets the SAGA object associated with exception
      Format:  get_message    (out object o);
      Inputs:  none
      Outputs: o:              the object associated with the
                               exception

+-------------------------------------------------------------+

Examples:
=========
```

```
  // c++ example
  int main ()
  {
    try
    {
      saga::file f ("file://localhost/etc/passwd");
      f.copy (/home/user/passwd.bak");
    }
    catch ( const saga::exception::PermissionDenied & e )
    {
      std::cerr << "SAGA error: No Permissions!" << std::endl;
    }
    catch ( const saga::exception & e )
    {
      std::cerr << "SAGA error: " << e.what () << std::endl;
    }

    return (0);
  }

+-------------------------------------------------------------+

Notes:
======

  There was discussion of using tagging interfaces, such as

  class LibraryException extends exception {
  }

  class LibraryFatalException implements-all LibraryException {
  }

  class LibraryRecoverableException
                              implements-all LibraryException {
  }

  class BackEndException extends exception {
  }

  class BackEndFatalException implements-all BackendException {
  }

  class BackendRecoverableException
                              implements-all BackendException {
  }

  In languages which allow exceptions, to allow a user of the
  API to additionally catch exceptions based upon the types,
  however such a tagging interface approach does not enforce
  that an exception falls into one of the types.
```

```
+-------------------------------------------------------------+
```

```
+--------------------------------------------------------------+

              #######
                 #       ##     ####   #    #
                 #      #  #   #       #   #
                 #     #    #   ####   ####
                 #     ######       #  #  #
                 #     #    #  #    #  #   #
                 #     #    #   ####   #    #

+--------------------------------------------------------------+
```

Summary:
========

Operations performed in widely distributed environments may take a long time to complete, and thus it is desirable to have the ability to perform operations in an asynchronous manner. There are many possible ways in which an asynchronous API may be developed --- the notes for this API contain several possibilities.

The main requirements the SAGA design team faced were ease of implementation in different languages, the ability to be implemented in a single-threaded environment, generality and ease of use.

This document defines an API and a pattern which associates a 'task' with each outstanding asynchronous operation. Each task represents an asynchronous version of one SAGA API method, and may have no one-to-one correspondence with any external process, such as a job.

```
+--------------------------------------------------------------+
```

Specification:
==============

```
  package saga.task
  {
    enum state
    {
        Unknown       = -1,
        New           =  1,
        Running       =  2,
        Done          =  3,
        Failed        =  4
    }
```

```
class task : implements-all saga::object
             implements-all saga::monitorable
{
  // no contructor
  DESCTRUCTOR   (in  Task           obj);

  run           (void);
  cancel        (void);
  wait          (in  double         timeout,
                 out boolean        finished);

  get_state      (out State          state);
  get_error      (out Exception      e);

  rethrow       (void);

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


class task_container : implements-all saga::object
                       implements-all saga::monitorable
{

  CONSCTRUCTOR  (out task_container  obj);
  DESCTRUCTOR   (in  task_container  obj);

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

  run           (void);
  cancel        (void);
  wait          (in  double         timeout,
                 int bool           all,
                 out Task           finished);

  get_tasks      (out array<Task,1>  tasks);
  get_states     (out array<State,1> states);

  // Metric:
  //     name:  state
  //     desc:  "fires if on changes of any task in container,
```

```
      //             and has the value of that tasks cookie."
      //     mode:  Read
      //     Unit:  1
      //     Type:  int
      //     Value: "0"
    }
  }

+-------------------------------------------------------------+


Details:
========

  // FIXME
  Each object in the SAGA API defines a create_task_factory method,
  which creates a corresponding factory object implementing the
  same set of methods as the original object, but returning a
  saga::task object.

  E.g. the saga::file class has a corresponding
  saga::directory_task_factory class, objects of which
  are instantiated by invoking directory.create_task_factory.  This
  directory_task_factory object has the same methods as those of
  the directory object; invoking any of these methods creates a
  task object representing an asynchronous call.

  Input and Output arguments of API calls MUST not be accessed
  or changed until the asynchronous task has completed; i.e.
  until 'wait' has been invoked on the task object and returned
  that the task state is Done or Failed.  Output values are only
  defined if the task is in Done state.

  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

    Running
      The run() method has been invoked on the task.

    Failed
      The asynchronous operation has unsuccessfully finished,
      or has been cancelled.
      This state is final.

    Done
      The asynchronous operation has successfully finished.
```

```
      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::ASync
    or saga::Task).

    - DESTRUCTOR
      Purpose: destroy the object
      Format:  DESTRUCTOR           (in  Task obj)
      Inputs:  obj:                  the object to destroy
      Outputs: none

    - run
      Purpose: Start the asynchronous operation.
      Format:  run ();
      Inputs:  none
      Outputs: none
      Throws:  IncorrectState

    - wait
      Purpose: Wait for the task to finish.
      Format:  wait                 (in  double  timeout,
                                     out boolean done);
      Inputs:  timeout:              number of seconds to wait
      Outputs: done:                 indicating if the task is
                                     done running
      Throws:  IncorrectState
      Notes:   - timeout < 0.0 wait forever
               - timeout = 0.0 return immediately
               - timeout > 0.0 wait for this number of seconds
               - a task must be in New, Running, Done or Failed
                 state for wait to succeed.
                 Otherwise an exception is thrown.
               - if the task is Running or New, and on
                 timout of wait is still Running or New,
                 false is returned
               - if the task is Running or New, and on
                 timout of wait is not Running or New
                 anymore, true is returned.  True hence
                 indicates that the task reached a final state.

    - cancel
      Purpose: Cancel the asynchronous operation.
      Format:  cancel ();
      Inputs:  none
      Outputs: none
      Notes:   - cancel can return immediately, even if the
```

```
                operation is neither done nor definitely
                cancelled.  The state remains 'Running' until
                the cancel operation succeeded - the state then
                changes to 'Failed'.
              - Information relying on the task state are
                undefined after calling cancel.

  - getState
    Purpose: Get the state of the task.
    Format:  getState             (out State state);
    Inputs:  none
    Outputs: state:                state of the task.

  - getError
    Purpose: Get the error of a failed task.
    Format:  getState             (out exception e);
    Inputs:  none
    Outputs: e:                    exception of task.

  - rethrow
    Purpose: re-throw any error a failed task catched.
    Format:  throw                ();
    Inputs:  none
    Outputs: none


class task_container:

  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 and other operations for a set of tasks.

  - add
    Purpose: Add a Task to a task_container.
    Format:  add                  (in Task task);
    Inputs:  task:                 task to add to the
                                   task_container
    Outputs:

  - remove
    Purpose: Remove a Task from a task_container.
    Format:  remove               (in Task task);
    Inputs:  task:                 task to remove from the
                                   task_container
    Outputs: none
    Throws:  DoesNotExist

  - run
    Purpose: Start all asynchronous operations in the
             container.
```

```
      Format:  run ();
      Inputs:  none
      Outputs: none
      Throws:  IncorrectState

    - wait
      Purpose: Wait for one or more of the tasks to finish.
      Format:  wait                 (in  double timeout,
                                     in  bool   all
                                     out Task   done);
      Inputs:  timeout:              number of seconds to wait
               all:                  if true wait for all tasks
      Outputs: done:                 finished task
      Throws:  IncorrectState
      Notes:   - < 0.0 wait forever
               - = 0.0 return immediately
               - > 0.0 wait for this number of seconds
               - for the exception condition, see the wait
                 description in the task class.
               - if 'all' is true, 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.

    - cancel
      Purpose: Cancel all the asynchronous operations in the
               container.
      Format:  cancel ();
      Inputs:  none
      Outputs: none
      Notes:   - see semantics of task cancel.

    - get_states
      Purpose: Get the states of all tasks in the task
               task_container.
      Format:  getStates            (out array<State,1> states);
      Outputs: States:               array of States for
                                     tasks in task_container

    - get_tasks
      Purpose: Get the tasks in the task task_container.
      Format:  get_tasks             (out array<Task,1> tasks);
      Outputs: tasks:                array of Tasks in
                                     task_container


+-------------------------------------------------------------+

Examples:
```

```
=========

  // c++ example, partly pseudocode
  saga::directory dir;
  saga::job       job;

  ...

  /* Create Tasks */
  saga::task t1 = dir.ls         <saga::task> (result);
  saga::task t2 = dir.copy       <saga::task> (source,target);
  saga::task t3 = dir.move       <saga::task> (source,target);
  saga::task t4 = job.checkpoint <saga::task> ();
  saga::task t5 = job.signal     <saga::task> (SIG_USR);

  // Start Tasks
  t1.run ();
  t2.run ();
  t3.run ();
  t4.run ();
  t5.run ();

  // put all tasks into container
  saga::task_container tc;

  tc.add (t1);
  tc.add (t2);
  tc.add (t3);
  tc.add (t4);
  tc.add (t5);

  // take one out again
  tc.remove (t5);

  // wait for all tasks in container to finish
  tc.wait ();

  // wait for the last task
  t5.wait ();

+-------------------------------------------------------------+


Notes:
======

  Error Handling:
  ===============
  {
    task.run  ();
    task.wait ();
```

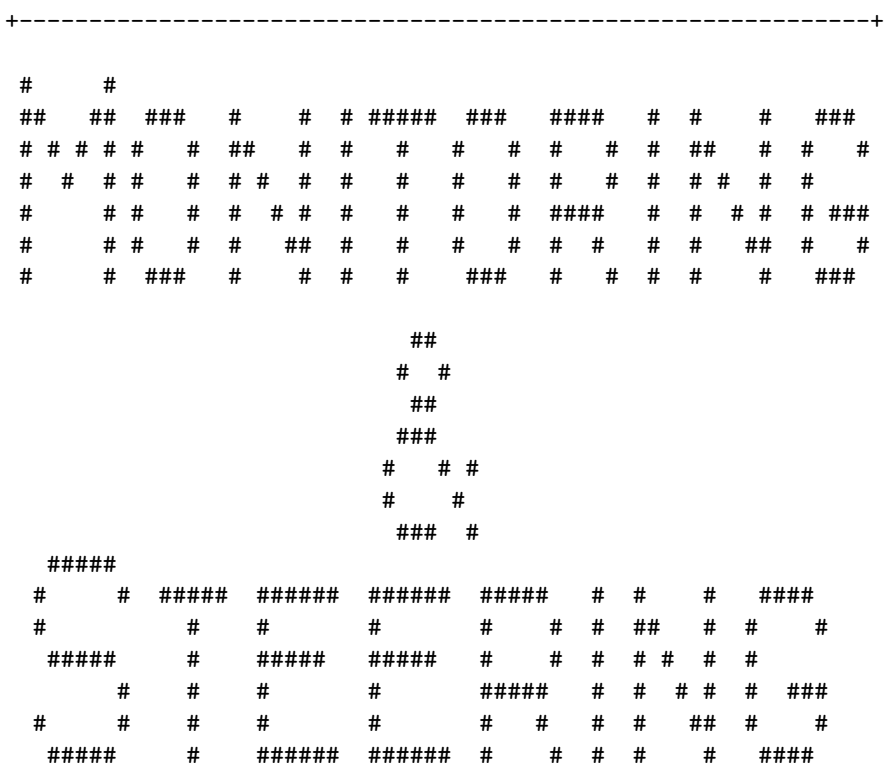
```
    if ( task.get_state = saga::task::Failed )
    {
      try {
        task.rethrow ();
      }
      catch ( saga::exception e )
      {
        std::cout << "task failed: " << e.what () << std::endl;
      }
    }
  }
```

```
  Task models:
  ============

  We had six different task models, as shown in example form
  below.  Model (E) has no compile-time sanity checking.  Model
  (F) allows only one asynchronous operation per object.  Once
  these models were eliminated, the choice between the remaining
  four was a matter of aesthetics as they all have equivalent
  functionality.

  The task container could have more methods to ease retrievel
  and manipulation of tasks.  E.g. the ability to label tasks
  and retrieve by label.

  ----------------------------
  Directory dir ("foo://bar/baz")
  Job       job = ...

  ----------------------------
  Model A)

  In this model there is a Task class associated with each API
  class, which is created by a create_task method.  Once a Task
  object has been created the asynchronous operation is invoked
  on it to associate an operation with the Task.

  /* Create Tasks */
  dir_task dt1 = dir.create_task ();
  dir_task dt2 = dir.create_task ();
  dir_task dt3 = dir.create_task ();
  job_task jt1 = job.create_task ();
  job_task jt2 = job.create_task ();

  /* Invoke operations on Task Objects */
  dt1.ls         ();
  dt2.copy       (source,target);
  dt3.move       (source,target);
```

```
jt1.checkpoint ();
jt2.signal     (USR);

/* Start Tasks */
dt1.run ();
dt2.run ();
dt3.run ();
jt1.run ();
jt2.run ();

-----------------------------
Model B)

In this model there is a task_factory class associated with each
API class, which is created by a create_task_factory method.
Once a task_factory object has been created the asynchronous
operation is invoked on it to create a Task object.

/* Create Task factories */
dir_task_factory dtf = dir.create_task_factory ();
job_task_factory jtf = job.create_task_factory ();

/* Create Tasks */
task t1 = dtf.ls         ();
task t2 = dtf.copy       (source,target);
task t3 = dtf.move       (source,target);
task t4 = jtf.checkpoint ();
task t5 = jtf.signal     (USR);

/* Start Tasks */
t1.run ();
t2.run ();
t3.run ();
t4.run ();
t5.run ();

saga::directory dir (url);
saga::directory::task_factory  dft = dir.get_task_factory ();

std::list <std::string> listing;

               dir.list (listing);
saga::task  t = dtf.list (listing);

t.run ();

-----------------------------
Model C)

In this model there is an object as an attribute on each API
object.  Invoking an operation on this object creates a Task.
```

```
/* Create Tasks */
task t1 = dir.task.ls         ();
task t2 = dir.task.copy       (source,target);
task t3 = dir.task.move       (source,target);
task t4 = job.task.checkpoint ();
task t5 = job.task.signal     (USR);

/* Start Tasks */
t1.run ();
t2.run ();
t3.run ();
t4.run ();
t5.run ();

-----------------------------
Model D)

In this model there is an equivalent for each API call which
creates an asynchronous task.

/* Create Tasks */
task t1 = dir.task_ls         ();
task t2 = dir.task_copy       (source,target);
task t3 = dir.task_move       (source,target);
task t4 = job.task_checkpoint ();
task t5 = job.task_signal     (USR);

/* Start Tasks */
t1.run ();
t2.run ();
t3.run ();
t4.run ();
t5.run ();

-----------------------------
Model E)

In this model, there is a get_task method associated with each
API object, which creates a Task given a string argument
defining the operation.

/* Create Tasks */
task t1 = dir.get_task ("ls");
task t2 = dir.get_task ("copy",source,target);
task t3 = dir.get_task ("move",source,target);
task t4 = job.get_task ("checkpoint");
task t5 = job.get_task ("signal",USR);

/* Start Tasks */
t1.run ();
```

```
  t2.run ();
  t3.run ();
  t4.run ();
  t5.run ();

  ------------------------------
  Model F)

  In this model, there is an asynchronous version of each API
  call, and each API class has a 'wait' method.  As there is no
  Task object, only one asynchronous operation may be outstanding
  on any object.

  dir.async_ls         ();
  job.async_checkpoint ();
  job.wait             ();
  dir.wait             ();
  dir.async_copy       (source,target);
  dir.wait             ();
  dir.async_move       (source,target);
  job.async_signal     (USR);
  job.wait             ();
  dir.wait             ();

+-------------------------------------------------------------+
```



Summary:
========

The ability to query Grid entities about state is requested in several SAGA use cases. Also, the SAGA Task model incorporates a certain amount of task 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 monitorables.

```
+-------------------------------------------------------------+


Specification:
==============

  package saga.monotoring
  {
    // a metric represents an entity / value to be monitored.
    class metric : implements-all saga::object
                   implements-all saga::attribute
    {
      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);

      // manage callbacks on the metric
      // add a callback, which gets active whenever
      // the metric changes (fires)
      add_callback       (in  call_back     * cb,
                          out int             cookie);

      // remove the callback
      remove_callback    (in  int             cookie,
                          out call_back     * cb);


      // push a new value to the consumers
      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' or 'ReadWrite'
    //
    //   name:  unit
    //   desc:  unit of metric
    //   mode:  ReadOnly
    //   type:  String
    //
    //   name:  type
    //   desc:  value type of metric
    //   mode:  ReadOnly
    //   type:  String
    //   value: 'Int', 'Float', 'Bool' or 'String'
    //
    //   name:  value
    //   desc:  value of metric
    //   mode:  depending on the mode attribute above
    //   type:  String
    //   value: formating restrictions described below apply
    //
  }


  // callbacks are used for asynchroneous notification of
  // metric changes (== events)
  interface call_back
  {
    callback         (in  metric          metric,
                      out bool            keep);
  }

  // SAGA objects which provide metrics and steering via
  // metrics, implement the monitorable interface
  interface monitorable
  {
    // introspection
    list_metrics       (out array<string,1> names);

    // get an availble metric for monitoring/steering
    get_metric         (in  string          name,
                        out metric          metric);

    // add an application hook for monitoring/steering
    add_metric         (in  metric          metric);

    // removes an application hook for monitoring/steering
    remove_metric      (in  string          name);

    // add a callback, which gets active whenever
    // the respective metric changes (fires)
    add_callback       (in  string          name,
```

```
                          in  call_back     * cb,
                          out int             cookie);

      // remove the callback
      remove_callback    (in  int             cookie,
                          out call_back     * cb);

      // steering: push the updated metric value (fire)
      // on a ReadWrite metric
      fire_metric        (in  string          name);
    }
  }

+-------------------------------------------------------------+


Details:
========

  class metric:
  -------------

    The fundamental object introduced in this package is a
    metric.  A metric representas an observable, which can be
    readable, writeable, or read/writable.  The availability of
    a readable observable corresponds to monitoring; 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", "Write", "ReadWrite" or "Final"
                 - ex:  "ReadWrite"
```

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

type:      "String", "Int", "Float" etc.
             - ex:   "Float"

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

The name of the metric must be unique, as it is used in most 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 describe the operation the metric descibes (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.progress       // applies to any operation on file
- job.state
- job.temperature     // a custom observable on a job
```

All attributes, apart from 'value', are ReadOnly, and are initialized in the metric constructor. If any other attribute than 'value' changes, a new metric needs to be created.

The following attribute values MUST be interpreted case insensitive: name, type, mode, unit.

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:  Bool
        //   value: True
      }
```

These specifications are NORMMATIVE, even if described as comments in the SIDL specification! The specified metrics MUST be supported by an implementation, unless noted otherwise, e.g.:

```
      //   note:  MAY be supported
```

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

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 DoesNotExist 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, and no other changes are allowed after that change.

```
- 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
  Notes:   - a metric is NOT attached to a session, but
             can be used for monitoring objects from
             different sessions.
           - the string arguments given are used to
```

```
              initialise the attributes of the metric, which
              are subsequently ReadOnly (see description
              above).

  - DESTRUCTOR
    Purpose: destroy the object
    Format:  DESTRUCTOR         (in  metric obj)
    Inputs:  obj:                the object to destroy
    Outputs: none

  // manage callbacks on the metric
  - add_callback
    Purpose: add asynchron notifier callback to watch metric
             changes
    Format:  add_callback       (in  call_back     * cb,
                                 out int              cookie);
    Inputs:  cb:                 callback class instance
    Outputs: cookie:             handle for this callback,
                                 to be used for removal
    Throws: IncorrectState
    Notes:   - IncorrectState is thrown if the metric is Final
             - the 'callback' method on cb will be invoked on
               any change of the metric value
             - if the 'callback' method returns true, the
               callback is kept registered; if it returns
               false, the callback is called only once, and
               is un-registered after completion.  Note that,
               due to threading, multiple cb instances can be
               active at the same time, so returning false can
               lead to race conditions.
             - the cb is passed by reference, not by value, so
               that no copy of private cb data is implied.
               The user has to ensure data consistency if a cb
               instance is used multiple times.

  - remove_callback
    Purpose: remove a callback from a metric
             changes
    Format:  remove_callback    (in  int             cookie,
                                 out call_back     * cb);
    Inputs:  cookie:             handle identifying the cb to
                                 be removed
    Outputs: cb:                 the removed cb reference
    Notes:   - if the callback was removed earlier, or
               was unregistered by returning false, no
               exception is thrown.  However, the returned cb
               reference is NULL.
             - the removal only affects the cb reference
               identified by 'cookie', even if the same
               reference was registered multiple times.
             - no assumption is made about threading: a
```

```
                returned cb can very well be active at the time
                of removal, e.g. can in a nother thread execute
                the callback method.  The user has to ensure
                the correct shutdown.


    - fire
      Purpose: push a new metric value to the backend
      Format:  fire               (void);
      Throws:  IncorrectState
      Notes:   - IncorrectState is thrown if the metric is not
                 Writable - so also if the metric was Writable,
                 but is meanwhile flagged Final!  To catch
                 race condition triggered exceptions, each file
                 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
                 explicitely called, no consumer will notice.
               - any callback registered on the metric in the
                 firing application is invoked on fire().
               -


  interface callback:
  -------------------

    - callback
      Purpose: asynchroneous handler for metric changes
      Format:  callback           (in  metric          metric,
                                   out bool            keep);
      Inputs:  metric:             the metric causing the
                                   callback invocation
      Outputs: keep:               indocates if cb stays
                                   registered
      Notes:   - see notes to the metric class above
               - 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.
               - if 'keep' is returned as false, all active
                 callback are finished normally, even if they
                 run longer than the instance returning 'false'.
                 However, it is guaranteed that no new
                 activation accurs after one instance returns
                 'false'.
```

```
              - 'metric' is the metric the callback is invoked
                on - that means that this metric recently
                changed its value.  Note that this change is
                semantically defined by the metric: the string
                of the 'value' attribute of the metric might
                have the same value in two subsequent
                invocations of the callback.
              - a callback can be added to a metric multiple
                times.
              - a callback can be added to multiple metrics at
                the same time.



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 notification callbacks to
  these metrics.

  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.

  The method add_metric() allows to implement steerable
  applications.  Metrics added with this method SHOULD be
  available on the saga job object representing the
  application.

  // introspection
  - list_metrics
    Purpose: list all metrics associated with the object
    Format:  list_metrics       (out array<string,1> names);
    Outputs: names:              array of names identifying
                                 the metrics associated with
                                 the object instance
    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.
             - no order is implied on the returned array
             - the returned array is guaranteed to have no
               double entries

  - 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:  DoesNotExist
    Notes:   - multiple calls of this method return multiple
               identical instances of the metric.


  - add_metric
    Purpose: add a metric instance to the application instance
    Format:  add_metric         (in  metric  metric);
    Inputs:  metric:             metric to be added
    Throws:  DoesAlreadyExist
    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)

  - remove_metric
    Purpose: remove a metric instance
    Format:  remove_metric      (in  string  name);
    Inputs:  name:               identifies metric to be
                                 removed
    Throws:  BadParameter
    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.

  - add_callback
    Purpose: add a callback to the specified metric
    Format:  add_callback       (in  string      name,
                                 in  call_back * cb,
                                 out int         cookie);
    Inputs:  name:               identifies metric to which cb
                                 is to be added
             cb:                 reference of callback class
                                 instance to be registered
    Outputs: cookie:             handle to be used for removal
                                 of the callback
    Throws:  DoesNotExist   - metric is unknown
    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:  DoesNotExist   - metric is unknown
      Notes:   - notes to the remove_callback method of the metric
                 class apply

    - fire_metric
      Purpose: push a new metric value to the backend
      Format:  fire_metric        (int  string  name);
      Inputs:  name:               identifies metric to be fired
      Throws:  DoesNotExist
               IncorrectState
      Notes:   - notes to the fire method of the metric
                 class apply


+-------------------------------------------------------------+


Examples:
=========

  callback example: trace all metric changes:
  -------------------------------------------

    // callback definition
    class trace_cb : public saga::callback
    {
      public:
        bool callback (saga::metric m)
        {
          std::cout << "metric " << m.get_attribute ("name")
                    << " fired." << std::endl;
          return true; // stay registered
        }
    }

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

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

      saga::job j = [...]; // details see description of saga::job
```

```
    j.add_metric ("task.state", &cb);

    ...
  }


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

  // c++ example for task state monitoring
  class write_metric_cb : public saga::callback
  {
   private:
     saga::task t_;

   public:
     write_metric_cb (const saga::task & t) { t_ = t; }

     bool callback (saga::metric & m)
     {
       std::cout << "bytes written: "
                 << m.get_attribute ("value")
                 << std::endl;

       std::cout << "task state:   "
                 << t_.t_state ()
                 << std::endl;

       return (false); // keep calback registered
     }
  };

  int main (int argc, char** argv)
  {
    ssize_t     len = 0;
    std::string str ("Hello SAGA\n");
    std::string url (argv[1]);

    saga::file   f (url);
    saga::task   t = f.write <saga::task> (str, &len);

    // assume that file has a 'progress' metric indicating
    // the number of bytes already written.  In general,
    // the list of metric names has to be searched for an
    // interesting metric, unless it is a default metric as
    // specified in the SAGA spec.

    // add the callback
    write_metric_callback cb (t);
    f.add_callback ("progress", &cb);
```

```
    // wait until task is done, and give cb chance to get
    // called a couple of times
    t.wait ();
  }


steering example: steer a remote job
------------------------------------

  // example for computaional steering (metric is writable).
  class observer_cb : public saga::metric::callback
  {
   private:
     saga::task t;

   public:
     bool callback (saga::metric & m)
     {
        int val = atoi ( m.get_attribute ("value") );

        std::cout << val << " is the new value." << std::endl;

        return (false); // keep callback registered
     }
  };

  // the steering appliciation
  int main (int argc, char** argv)
  {
    saga::job_service js;

    saga::job j = js.run ("remote.host.net",
                          "my_remote_application");

    // Assume that job has a 'param_1' metric representing
    // a integer parameter for the remote application.
    // First add observer metric - that causes the
    // asynchroneous printout of any changes to the value
    // of the 'param_1' metric
    observer_cb cb;
    j.add_callback ("param_1", &cb);

    // then get metric for active steering
    saga::metric m = j.get_metric ("param_1");

    for ( int i = 0; i < 10; i++ )
    {
      // if param_1 is ReadOnly, set_value would throw
      // 'IncorrectState'
      m.set_attribute ("value", std::string (i));
```

```
      // push the pending change out to the receiver
      m.fire ();

      // callback should get called NOW + 2*latency
      // That means fire REQUESTS the value change, but only
      // the remote job can CHANGE the value - that change
      // needs then reporting back to us.

      // give steered application some time to react
      sleep (1);
    }
  }



steering example: BE a steerable job
------------------------------------

the example shows a job which
  - creates a metric to expose a Float steerable parameter
  - on each change of that parameter computes a new isosurface

  // callback - on any change of the metric value, e.g. due to
  // steering from a remote GUI application, a new iso surface
  // is computed
  class my_cb : public saga::callback
  {
    public:
      // the callback gets called on any
      bool callback (saga::metric m)
      {
        // get the new iso-value
        float iso = atof (m.get_attribute ("value"));

        // compute an isosurface with that iso-value
        compute_iso (iso);

        // keep this callback alive, and get called again on
        // the next metric event.
        return (false);
      }
   }

  int main ()
  {
    // create a metric for the iso-value of an isosurfacer
    saga::metric m ("application.isosurfacer.isovalue",
                    "iso-value of the isosurfacer",
                    "ReadWrite",
                    "",
```

```
                    "Float",
                    "1.0");

    // add the callback which reacts on changes of the
    // metric's value (cookie is ignored)
    my_cb cb;
    m.add_callback (&cb);

    // get job handle for myself
    saga::job_server js;
    saga::job self = js.get_self ();

    // add metric to job
    self.add_metric (m);

    // now others can 'see' the metric, e.g. via
    // job.list_metrics ();

    // now, the callback could also have been added with:
    // self.add_metric ("application.isosurfacer.isovalue", cb);

    // compute isosurfaces for the next 10 minutes -
    // the real work is done in the callback
    sleep (600);

    // on object (self) destruction, metrics and callback
    // objects are destroyed as well
    return (0);
  }
```

monitoring example: callback for stream connects

```
  // my callback container class
  class my_cb : public saga::callback
  {
    privat:
      // we keep a stream server and a single client stream
      saga::stream_server ss_;
      saga::stream        s_;

    public:
      // constructor initialises these (note that the
      // client stream should be not connected at this
      // point
      my_cb (saga::stream_server ss,
             saga::stream        s )
      {
        ss_ = ss;
```

```
        s_  = s;
      }

    ~my_cb (void) { }

      // the callback gets called on any
      bool callback (saga::metric m)
      {
        // the stream server got an event triggered, and
        // should be able to create a client socket now.
        s_ = ss_.wait ();

        if ( s_.state == saga::stream::open )
        {
          // have a client stream, we are done
          // don't call this cb again!
          return (true);
        }

        // no valid client stream obtained: keep this
        // callback alive, and get called again on the
        // next metric event.
        return (false);
      }
  }

  int main ()
  {
    // create a stream server, and an not connected
    // stream
    saga::stream_server ss;
    saga::stream        s;

    // give both to our callback class, and register that
    // callback with the 'client_connect' metric of the
    // server.  That causes the callback to be invoked on
    // every change of that metric, i.e. on every event
    // that changes that metric, i.e. on every client
    // connect attempt.
    my_cb cb (ss, s);
    ss.add_callback ("client_connect", &cb);

    // now we serve incoming clients forever
    while ( true )
    {
      // check if a new client is connected
      // the stream state would then be Open
      if ( s.state == saga::stream::Open )
      {
        // a client got conncted!
        // handle open socket
```

```
          s.write ("You say hello, I say good bye!\r\n", 32);

          // and close stream
          s.close ();

          // the stream is not Open anymore.  We re-add the
          // callback, and hence wait for the next client
          // to connect.
          ss.add_callback ("client_connect", &cb);
        }
        else
        {
          // no client yet, idle, or do something useful
          sleep (1);
        }
      }

      // we should never get here
      return (-1);
    }


+-------------------------------------------------------------+

Notes:
======

  - possible deviation: allow only one CB per metric:
    no add/remove, but set/reset CB

  - other useful attributes might be:

    - update-mode (discreet, cont, static)
      This attribute would describe how often an attached
      callback gets activated.
      discreet: on changes of the metric value
                example: job state (value is always defined, but
                changes infrequently)
      cont:     as fast as possible
                example: time (value always defined, and changes
                continously)
      static:   once
                example: complete (value changes once)

    - value-range (1-10, [1,2,3], ...)

      That attribute obviously depends on type and unit of
      the metric, and is therefore not useful unless types
      attribute values or typed metric values are introduced in
      SAGA.
```

```
      In particular computational steering would benefit from
      value ranges, as that allows client side range checks, and
      the creation of sensible client side user interfaces.


+--------------------------------------------------------------+
```

```
+--------------------------------------------------------------+

     #
    # #   ##### ##### #####   #  #####  #    # ##### ######
   #   #    #     #   #    #  #  #    # #    #   #   #
  #     #   #     #   #    #  #  #####  #    #   #   #####
  #######   #     #   #####   #  #    # #    #   #   #
  #     #   #     #   #   #   #  #    # #    #   #   #
  #     #   #     #   #    #  #  #####   ####    #   ######

+--------------------------------------------------------------+

Summary:
========

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

  Objects implementing this interface can 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 types in later
  versions of this API specification.

  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 CAN
  issue a warning if a specified attribute cannot be supported.

  Implementations are strongly discouraged to include other,
  implementation specific attributes in SAGA, as that would bind
  applications to that specific implementation, and limit
  portability, which is a declared goal of the SAGA approach.

+--------------------------------------------------------------+

Specification:
==============

  package saga.atrribute
  {
    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,1> values);
      get_vector_attribute    (in  string          key,
                               out array<string,1> values);
      remove_attribute        (in  string          key);

      // inspection methods
      list_attributes         (out array<string,1> keys);
      equals                  (in  string          key,
                               in  string          val,
                               out bool            test);
      has_attribute           (in  string          key,
                               out bool            test);
      has_vector_attribute    (in  string          key,
                               out bool            test);
      is_readonly             (in  string          key,
                               out bool            test);
      is_writable             (in  string          key,
                               out bool            test);
    }
  }
```

```
+-------------------------------------------------------------+
```

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 limited to a finite and well defined set of keys, unless otherwise specified.

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.

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

In order to limit semantic and syntactic ambiguities (e.g. due to spelling deviations), language bindings SHOULD 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.

```
  val 1: home, sweet home
  val 2: Global GF
  val 3: SAGA
  string: "home\, sweet home, Global 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 ignored, and CAN be changed by the SAGA implementation. The equals method does also not rely on ordering (i.e. '"one" "two"' equals '"two" "one"').

Values of boolean type attributes MUST be expressed as one of the following: '0', '1', 'true', 'false', 'TRUE', 'FALSE'.

Values of floating point type attributes are expressed as they would result in a printf of the format "%lld", as defined by POSIX.

Values of integer type attributes are expressed as they would result in a printf of the format "%Lf", as defined by POSIX.

Values of string type attributes are expressed as-is, however, comma, backslashes and leading spaces are escaped by a backslash, as described above.

Some of the means above are aimed at the prevention of abuse of the attribute interface for implementation specific extensions and semantic overloading. However, we think that the attribute interface is flexible enough to accommodate small changes to the SAGA API in future versions of this specification with minor or no code changes, as long as the attribute interface as described below is used - both SAGA implementors and users are hence couraged to prefer that interface to thee convenience declarattions described above.

```
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: none
    Throws:  ReadOnly
    Notes:   - a value of NULL means to remove the attribute
             - a empty string means to set an empty value
               (the attribute remains present).


  - 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:  DoesNotExist


  - set_vector_attribute
    Purpose: set an attribute to an array of values.
    Format:  set_vector_attribute (in string          key,
                                   in array<string,1> values);
    Inputs:  key:                  attribute key
             values:               array of values for the
                                   attribute
    Outputs: none
    Throws:  ReadOnly


  - get_vector_attribute
    Purpose: get the array of values associated with an
             attribute
    Format:  get_vector_attribute (in string           key,
                                   out array<string,1> values);
    Inputs:  key:                  attribute key
    Outputs: values:               array of values of the
                                   attribute.
    Throws:  DoesNotExist


  - remove_attribute
```

```
    Purpose: removes an attribute completely.
    Format:  remove_attribute     (in  string key)
    Inputs:  key:                  attribute key
    Outputs: none
    Throws:  DoesNotExist
             ReadOnly
    Notes:   - Any following query or set operation will not
               find the attribute present anymore.


  - list_attributes
    Purpose: Get the list of attribute keys.
    Format:  list_attributes      (out array<string,1> keys);
    Inputs:  none
    Outputs: keys:                 existing attribute keys


  - equals
    Purpose:
    Format:  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
    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 (order of the
               elements is ignored).


  - has_attribute
    Purpose:
    Format:  has_atttribute       (in  string key,
                                   out bool   test);
    Inputs:  key:                  attribute key
    Outputs: test                  bool indicating success
    Notes:   - This method returns TRUE if the attribute
               identified by key exists and is a scalar
               attribute.
             - It does NOT throw a DoesNotExist exception.


  - has_vector_attribute
    Purpose:
    Format:  has_vector_attribute (in  string key,
                                   out bool   test);
```

```
      Inputs:  key:                  attribute key
      Outputs: test                  bool indicating success
      Throws:
      Notes:   - This method returns TRUE if the attribute
                 identified by key exists and is a vector
                 attribute.
               - It does NOT throw a DoesNotExist exception.


    - is_readonly
      Purpose:
      Format:  is_readonly          (in  string key,
                                     out bool   test);
      Inputs:  key:                  attribute key
      Outputs: test                  bool indicating success
      Throws:  DoesNotExist
      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 or
                 set_vector_attribute.


    - is_writable
      Purpose:
      Format:  is_writable          (in  string key,
                                     out bool   test);
      Inputs:  key:                  attribute key
      Outputs: test                  bool indicating success
      Throws:  DoesNotExist
      Notes:   - This method returns TRUE if the attribute
                 identified by the key exists, and can be
                 changed by set_attribute or
                 set_vector_attribute.

+-------------------------------------------------------------+

Examples:
=========

  // c++ example:
  job_definition d;

  std::list <char*> env;
  env.push_back ("a = b");
  env.push_back ("c = d");

  d.set_attribute        ("JobCmd", "/bin/ls");
  d.set_vector_attribute ("JobEnv", env);

+-------------------------------------------------------------+
```

```
Notes:
======

  A find on attributes (both keys and values) should be
  considered, as that is needed and defined on other classes
  (saga::locical_file, saga::advert_service) anyway.

+--------------------------------------------------------------+
```

```
+-------------------------------------------------------------+

                #     #
                ##    #    ##    #    #  ######
                # #   #   #  #   ##  ##  #
                #  #  #  #    #  # ## #  #####
                #   # #  ######  #    #  #
                #    ##  #    #  #    #  #
                #     #  #    #  #    #  ######

             #####
            #     #  #####     ##     ####   ######
            #        #    #   #  #   #    #  #
             #####   #    #  #    #  #       #####
                  #  #####   ######  #       #
            #     #  #       #    #  #    #  #
             #####   #       #    #   ####   ######

+-------------------------------------------------------------+
```

Summary:
========

This file describes interfaces which operate on arbitrary hierarchical namespaces, such as those used in physical, virtual and logical file systems, and information systems.

Several SAGA packages share the notion of namespaces and operations on these namespaces. In order to increase consistency in the API, those packages should share the same API paradigm.

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 interface included here deviates significantly from the POSIX version; this interface 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.

```
+-------------------------------------------------------------+
```

```
Specification:
==============

  package saga.name_space
  {
    enum flags
    {
      Unknown         =  -1,
      None            =   0,
      Overwrite       =   1,
      Recursive       =   2,
      FollowSymbolic  =   4,
      Create          =   8,
      Excl            =  16,
      Lock            =  32,
      CreateParents   =  64,
      DeReference     = 128
    }

    class ns_entry : implements_all saga::object
                     implements-all saga::monitorable
    {
      CONSTRUCTOR     (in  string             url,
                       in  array<flags,1>     flags,
                       in  session            session );
                       out ns_entry           obj     );
      DESTRUCTOR      (in  ns_entry           obj     );

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

      // navigation/query methods
      read_link       (out string             link    );
      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    );

      // management methods
      copy            (in  string             target,
                       in  array<flags,1>     flags   );
      link            (in  string             target,
                       in  array<flags,1>     flags   );
      move            (in  string             target,
                       in  array<flags,1>     flags   );
```

```
    remove          (void                          );
    close           (void                          );

    // helper methods
    convert         (in  string            template,
                     out string            new_url);
  }


  class ns_directory : implements-all saga::object
                       implements-all saga::monitorable
                       extends        saga::ns_entry
  {
    CONSTRUCTOR     (in  string            url,
                     in  array<flags,1>    flags,
                     in  session           session );
                     out ns_directory      obj     );
    DESTRUCTOR      (in  ns_directory      obj     );

    // navigation/query methods
    change_dir      (in  string            dir     );
    list            (in string             name,
                     out array<string,1>   names   );
    find            (in string             pattern,
                     in  array<flags,1>    flags,
                     out array<string,1>   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    );

    // management methods
    copy            (in  string            source,
                     in  string            target,
                     in  array<flags,1>    flags   );
    link            (in  string            source,
                     in  string            target,
```

```
                      in  array<flags,1>     flags   );
     move            (in  string             source,
                      in  string             target,
                      in  array<flags,1>     flags   );
     remove          (in  string             target,
                      in  array<flags,1>     flags   );
     touch           (in  string             target  );
     make_dir        (in  string             target,
                      in  array<flags,1>     flags   );

     // factory methods
     open            (in  string             name,
                      in  array<flags,1>     flags,
                      out ns_entry           entry   );
     open_dir         (in  string            name,
                      in  array<flags,1>     flags,
                      out ns_directory       dir     );
   }
  }

+-------------------------------------------------------------+


Details:
========

  Definitions:

   pathname:

     A 'Pathname' as accepted by this specification MUST follow
     the specification of pathnames as described in section
     1.1.3 "Pathnames" of the Document "Namespace Service" of
     the Grid File System Working Group (GFS-WG) in GGF [1].


   directory:

     A 'Directory' represent what [1] defines as 'Virtual
     Directories'.


   directory_entry:

     A 'directory_entry' or 'Entry' represent what [1] defines
     as 'Junction'.  Note that any type of junction defined
     there could be used.


  Wildcards:
```

```
    The API supports wildcards where appropriate.  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 strings where they can be used
    in the respective shell commands, such as:

      copy   *.txt dir
      move   *.txt dir
      link   *.txt dir
      ls     *.txt
      remove *.txt

    Users are rarely aware that wildcards can be used in
    unorthodox places, such as:

      move *.txt dir*
      move *

    The result of such operations is dependend on the order the
    wildcard expansion is performed, e.g. if 'dir*' expands to
    'dir_1 dir_2', all txt files and dir_1 will end up in dir_2.

    SAGA implementation MUST support wildcards for all strings
    where that ambiguity cannot arise, (source for move etc),
    and MAY support wildcards at all respective string
    parameters which would accepts wildcards in the respective
    POSIX shell calls.

    For the method calls on 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.  Flags MUST be
    applied to all elements of a wildcard expansion, even if
    that raises an exception for any reasons.


  interface ns_directory

    ns_directory defines two sets of methods: one set to navigate in
    the namespace hierarchy (e.g. cd, ls, find, ...), and one
    set to handle entities in the namespace (e.g. copy(),
    move(), open(), ...)
```

```
Methods for creating / destroying the ns_directory object

- CONSTRUCTOR
  Purpose: create the object
  Format:  CONSTRUCTOR          (in  string  url,
                                 in  array<flags,1> flags,
                                 in  Session session,
                                 out ns_directory  obj)
  Inputs:  url:                  initial working dir
           flags:                open mode
           session:              session handle for
                                 object creation
  Outputs: obj:                  the newly created object
  Notes:   - the session handle defaults to the SAGA
             default session handle if not explicitely
             specified
           - the default flag set is (None)

- DESTRUCTOR
  Purpose: destroy the object
  Format:  DESTRUCTOR           (in  ns_directory obj)
  Inputs:  obj:                  the object to destroy
  Outputs: none


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: none
  Throws:  BadParameter
           DoesNotExist
           IncorrectState
  Notes:   - dir can be relative to pwd
           - similar to the 'cd' command in Unix shells,
             as defined by POSIX


- list
  Purpose: list entries in this directory
  Format:  list               (in string           name,
                               out array<string,1> names);
  Inputs:  name:               name to list
  Outputs: names:              array of names existing in
                               the directory
  Throws:  BadParameter
           DoesNotExist
           IncorrectState
```

```
      Notes:   - similar to 'ls' in Unix shells, as defined
                 by POSIX
               - name can be relative to pwd
               - if name is NULL, the current directory contents
                 is listed.

    - find
      Purpose: find entries in this directory and below
      Format:  find               (in string           pattern,
                                   in  array<flags,1>  flags,
                                   out array<string,1> 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:  BadParameter
               IncorrectState
      Notes:   - similar to 'find' in Unix shells, as defined
                 by POSIX, but limited to the -name option.
               - pattern MUST not include a path element
               - 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 FollowSymbolic flag
                 is specified
               - find does list symbolically linkes entries
                 with matching name
               - the pattern follows the standard unix shell
                 wildcard specification, as defined by POSIX
               - the matching entries returned are complete
                 relative (to cwd) path names.
               - default flags set is 'Recursive' (1)

    - 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:  BadParameter
               DoesNotExist
               IncorrectState
      Notes:   - link may be relative or absolute depending on
                 underlying implementation.  However, the
                 returned name MUST be sufficient to access
                 the target entry
               - resolves one link level only
               - name can be relative to pwd
```

```
            - inspired by 'ls -L' command in Unix shells, as
              defined by POSIX

  - 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:  BadParameter
             IncorrectState
    Notes:   - name can be relative to pwd
             - as in 'test -e' in Unix shells, as defined
               by POSIX

  - is_dir
    Purpose: tests name for beeing a directory
    Format: is_dir              (in  string  name,
                                 in  int     flags,
                                 out boolean test);
    Inputs:  name:               name to be tested
             flags:              flags for operation
    Outputs: test:               boolean indicating if name
                                 is a directory
    Throws:  BadParameter
             DoesNotExist
             IncorrectState
    Notes:   - returns true if entry is a directory, false
               otherwise
             - name can be relative to pwd
             - flag can be set to DeReference, default is
               None
             - as in 'test -d' in Unix shells, as defined
               by POSIX

  - is_entry
    Purpose: tests name for beeing a ns_entry
    Format:  is_entry           (in  string  name,
                                 in  int     flags,
                                 out boolean test);
    Inputs:  name:               name to be tested
             flags:              flags for operation
    Outputs: test:               boolean indicating if name
                                 is a ns_entry
    Throws:  BadParameter
             DoesNotExist
             IncorrectState
    Notes:   - returns true if the entry is a entry, false
               otherwise (although a ns_dir IS_A ns_entry,
```

```
              false is returned on a test on a ns_dir)
            - name can be relative to pwd
            - flag can be set to DeReference, default is
              None
            - as in 'test -f' in Unix shells, as defined
              by POSIX

  - is_link
    Purpose: tests name for beeing a link
    Format:  is_link            (in  string  name,
                                 in  int     flags,
                                 out boolean test);
    Inputs:  name:               name to be tested
             flags:              flags for operation
    Outputs: test:               boolean indicating if name
                                 is a link
    Throws:  BadParameter
             DoesNotExist
             IncorrectState
    Notes:   - returns true if the entry is a link, false
               otherwise
             - name can be relative to pwd
             - flag can be set to DeReference, default is
               None
             - as in 'test -l' Unix shells, as defined
               by POSIX

  - get_num_entries
    Purpose: gives the number of entries in the directory
    Format:  get_num_entries    (out int   num);
    Inputs:  none
    Outputs: num:                number of entries in the
                                 directory
    Throws:  IncorrectState
    Notes:   - can be used for iteration through large
               directories (see get_entry)
             - at the time of using the result of this call,
               the actual number of entries may already have
               changed (no locking is implied)

  - 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
    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
              if there is no locking or state tracking in
              the underlying implementation
            - can be used for iteration through large
            directories


    Methods for operation on namespace entities:
    - copy
      Purpose: copy the entry to another part of the namespace
      Format:  copy               (in  string         source,
                                   in  string         target,
                                   in  array<flags,1> flags);
      Inputs:  source:             name to copy
               target:             name to copy to
               flags:              flags defining the operation
                                   modus
      Outputs: none
      Throws:  BadParameter
               DoesNotExist
               IncorrectState
      Notes:   - if the target is a directory the source entry
                 is copied into the directory
               - it is an error if the source is a directory
                 and the 'Recursive' flag is not set.
               - if the target already exists, it will be
                 overwritten if the 'Overwrite' flag is set,
                 otherwise it is an error
               - default flags set is empty (0)
               - both source and target can be relative to pwd
               - similar to the 'cp' command in Unix shells,
                 as defined by POSIX

    - 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  array<flags,1> flags);
      Inputs:  source:             name to link
               target:             name to link to
               flags:              flags defining the operation
                                   modus
      Outputs: none
      Throws:  BadParameter
               DoesNotExist
               IncorrectState
      Notes:   - if the target is a directory the source entry
```

```
              is linked into the directory.
            - if the target already exists, it will be
              overwritten if the 'Overwrite' flag is set,
              otherwise it is an error
            - default flag set is empty (0)
            - both source and target can be relative to pwd
            - similar to the 'ln -s' command in Unix shells,
              as defined by POSIX

  - 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  array<flags,1> flags);
    Inputs:  source:             name to move
             target:             name to move to
             flags:              flags defining the operation
                                 modus
    Outputs: none
    Throws:  BadParameter
             DoesNotExist
             IncorrectState
    Notes:   - if the target already exists, it will be
               overwritten if the 'Overwrite' flag is set,
               otherwise it is an error
             - moving '.' is not allowed, and throws
               a BadParameter exception
             - default flag set is empty (0)
             - both source and target can be relative to pwd
             - similar to the 'mv' command in Unix shells, as
               defined by POSIX

  - remove
    Purpose: removes the entry
    Format:  remove             (in  string         target,
                                 in  array<flags,1> flags);
    Inputs:  target:             entry to be removed
    Outputs: none
    Throws:  BadParameter
             DoesNotExist
             IncorrectState
    Notes:   - if the entry is a directory the 'Recursive'
               flag MUST be set or an exception will be
               raised
             - default flag set is empty (0)
             - removing '.' is not allowed, and throws
               a BadParameter exception
             - target can be relative to pwd
             - similar to the 'rm' command in unix shells,
               as defined by POSIX
```

```
- close
  Purpose: closes the object
  Format:  close              (void);
  Inputs:  none
  Outputs: none
  Throws:  IncorrectState
  Notes:   - IncorrectState is thrown if the object was
             closed before
           - any subsequent method call on the object
             MUST also raise IncorrectState (apart from
             DESTRUCTOR)

- touch
  Purpose: creates a new entry, or updates access time
  Format:  touch              (in  string target);
  Inputs:  target:             target to create
  Ouputs:  none
  Throws:  BadParameter
           IncorrectState
  Notes:   - target can be relative to pwd
           - similar to the 'touch' (1) call, as defined
             by POSIX
           - the target is created as ns_entry if it does
             not exist
           - the last access time of the target MUST be
             updated if the target exists
           - if the target string is NULL, the cwd is
             touched

- make_dir
  Purpose: creates a new directory
  Format:  make_dir           (in  string         target,
                               in  array<flags,1> flags);
  Inputs:  target:             directory to create
  Ouputs:  none
  Throws:  BadParameter
           AlreadyExists
           IncorrectState
  Notes:   - if the parent directory or directories do not
             exist, 'CreateParents' flag MUST be set or an
             exception will be raised.  If set, the parrent
             directories are created as well
           - an exception MUST be raised if the directory
             already exists
           - default flag set is empty (0)
           - target can be relative to pwd
           - similar to the 'mkdir' (2) call, as defined by
             POSIX

- open_dir
```

```
    Purpose: creates a new ns_directory instance
    Format:  open_dir           (in  string         name,
                                 in  array<flags,1> flags,
                                 out ns_directory   dir);
    Inputs:  name:               directory to open
             flags:              flags defining the operation
                                 modus
    Outputs: dir:                opened directory instance
    Throws:  BadParameter
             DoesNotExist
             IncorrectState
    Notes:   - the cwd of the new dir object instance is set
               to <name>
             - target can be relative to pwd
             - similar to the 'opendir' (3) call in Unix, as
               defined by POSIX


  - open
    Purpose: creates a new ns_entry instance
    Format:  open               (in  string         name,
                                 in  array<flags,1> flags,
                                 out ns_entry       entry);
    Inputs:  name:               entry
             flags:              flags defining the operation
                                 modus
    Outputs: entry:              opened entry instance
    Throws:  BadParameter
             DoesNotExist
             IncorrectState
    Notes:   - target can be relative to pwd
             - similar to the 'open' (2) call in Unix, as
               defined by POSIX


  - convert
    Purpose: converts url into a different name space
    Format:  convert            (in  string  template,
                                 out string  new_url);
    Inputs:  template:           template for new url
    Outputs: new_url:            translated url
    Notes:   - the method translates a url into a different
               URL which MUST point at the same endpoint
             - the form of the new URL is determined by
               the given template
             - the template gives the part of the new url
               which is known.
             - the template has to present a complete URL
               _beginning_, e.g.  "http://" or
               "http://host.net:123/", but not "host.net".
             - if convert cannot determine a url in the name
```

```
                space given by the template which points to the
                same endpoint, an empty string is returned.

+-------------------------------------------------------------+

Examples:
=========

  The interfaces are not implemented directly - for more
  examples, check out the physical and logical file
  specifications.

  Example: provide recursive directory listing for a given
           directory

  Note:    - check for '.' and '..' resursion are left as an
             exercise to the reader...
           - string operations and printf statements are
             obviously simplified...

   ---------------------------------------------------------------
   // c++ example
   std::string indent (int indent)
   {
     std::string s = "  ";

     for (int i = 0; i < indent; i++, s += "  ");

     return (s);
   }

   void list_dir (std::string & url,
                  int           indent = 0)
   {
     try
     {
       // create directory and iterate over entries
       saga::ns_dir dir (url);

       printf ("\n%s ---> %s\n", indent (indent), url);

       for ( int i = 0; i < dir.get_num_entries (); i++ )
       {
         char   type = '?';
         string info = "";

         // get name of next entry
         string name = dir.get_entry (i);

         // get type and other infos
         if ( dir.is_link (name) )
```

```
        {
          if (dir.exists(dir.read_link (name))){info="---> ";}
          else                                 {info="-|-> ";}
          info += dir.read_link (name);
          type  = 'l';
        }
        else if (dir.is_entry(name)){ type = 'f';            }
        else if (dir.is_dir  (name)){ type = 'd'; info = "/";}

        printf ("%s > %3d - %s - %s%s\n", indent (indent), i+1,
                                          type, name, info);

        // recursion on directories
        if ( dir.is_dir (name) )
        {
          list_dir (name, indent++);
        }
      }

      printf ("\n%s <--- %s\n", indent (indent), url);
    }

    // catch all errors - see elsewhere for better examples
    // of error handling in SAGA
    catch ( const saga::exception & e )
    {
      std::cerr << "Oops! SAGA error: "
                << e.what () << std::endl;
    }

    return;
  }
  ----------------------------------------------------------------

+--------------------------------------------------------------+


References:
===========

  [1] http://forge.gridforum.org/projects/gfs-wg/document/ \
            RNS-Proposed_Final_Draft-v1.10/en/10


+--------------------------------------------------------------+


Notes, Issues and Known Limitations:
====================================

  A useful extension to the presented interface is a find like
```

```
method.  However, the flags and options to find (1), are
manyfold, and it currently it is unclear how a good mapping to
an _simple_ SAGA API call might look like.

An Directory can be seen as a container of directory_entries,
which can be Files, Links, Directories etc.  That notion is not
reflected in this version of the interface, since no call is
taking such entities as arguemt, or is returning such entities.
However, a later version of this interface may introduce  this
distinction if necessary - it needs then to be reflected in all
classes implementing this interface.

In the current version, it is not possible to (e.g.) copy Files
w/o creating a directory first.  That seems in particular
cumbersome if the source and target namespace of the file copy
are different.  However, we think that the presented approach
is more coherent than the alternatives.

Similarily, 'stat' like calls seem (semantically) to specific
to the specific namespace incarnation to get included in this
rather generic specification.

The notion of security, permissions, ACLs, ownership etc. is
missing from this version of the spec, but is crucial to it's
usability ans acception.  It will get added as soon as there is
an agreement on security in the SAGA API in general.

The URL Problem (see intro):
----------------------------

  In order to settle the issue for the SAGA-CORE-API (NOT
  forever), we propose the following solution:

    - A URL utility class seems useful, but would increase
      the syntactic load on the current set of methods (either
      allow all methods with strings _and_ urls, or require
      transformation from strings to urls _always_ - both
      seems suboptimal).

    - hence NO URL utility class is mandated.

    - a NSEntry utility call is introduced:

        convert (in  string target,
                 out string new_url);

      Usage examples:

      saga::file  f ("ftp://ftp.net.org/pub/data/file.txt");
      std::string n1 = f.convert ("ftp://ftp.net.org:123/");
      std::string n2 = f.convert ("ftp://alias.net.org/data/");
```

```
        std::string n3 = f.convert ("http://www.net.org");

      - that call tries to translate URLs, and if unsuccessfull
        returns an empty string.

    I know that is 'somewhat' of a compromise.  A full fledged
    URL class certainly would be useful, and Java for example
    already has it.  I would expect that most SAGA
    implementations will provide one anyway.  But as we could
    not reach consensus on that, the topic seems to need more
    discussion, and more time which I wthink we don't have for
    stabilizing the API.


+--------------------------------------------------------------+
```

```
+--------------------------------------------------------------+

               #######
               #       #    #       ######
               #       #    #       #
               #####   #    #       #####
               #       #    #       #
               #       #    #       #
               #       #    ######  ######

+--------------------------------------------------------------+
```

Summary:
========

The ability to access files regardless of their location is central to many of the SAGA use cases (see below). This interface addresses the most common operations detailed in the use cases.

The interfaces are syntacically and semantically POSIX oriented, but also borrow some ideas from the GridFTP specification, which is nowadays widely used for remote data access.

Please note that the interactions with files as opaque entities (as entries in file name spaces) are covered by the name_space package. The interfaces presented here supplement the namespace package with operations for the reading and writing of files.

```
+--------------------------------------------------------------+
```

Specification:
==============

```
  package saga.file
  {
    enum flags
    {
      Unknown         =   -1, // same as in name_space::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
      FollowSymbolic  =    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
    DeReference     =  128, // same as in name_space::flags
    Truncate        =  256,
    Append          =  512,
    Read            = 1024,
    Write           = 2048,
    ReadWrite       = 4096,
    Binary          = 8192
  }

  enum seek_mode
  {
    Unknown    =  -1,
    Start      =   1,
    Current    =   2,
    End        =   3
  }

  struct ivec
  {
    long          offset;  // position of data  to be r/w
    long          length;  // number   of bytes to be r/w
    array<byte,1> buffer;  // data             to be r/w
    long          result;  // number   of bytes       r/w
  }


  class directory : implements-all saga::object
                    implements-all saga::monitorable
                    extends        saga::ns_directory
  {

    CONSTRUCTOR (in  string                 url,
                 in  session                session,
                 in  int                    flags = None,
                 out directory              dir      );
    DESTRUCTOR  (in  directory              dir      );

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

    open_dir    (in  string                 name,
                 in  int                    flags = None,
                 out directory              dir      );

    open        (in  string                 name,
                 in  int                    flags = Read,
```

```
                out file                    file     );
}


class file : implements-all saga::object
             implements-all saga::monitorable
             extends        saga::ns_entry
{

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

  read        (in    long                    len_in,
               inout array<byte,1>           buffer,
               out   long                    len_out  );
  write       (in    long                    len_in,
               in    array<byte,1>           buffer,
               out   long                    len_out  );
  seek        (in    long                    offset,
               in    seek_mode               whence,
               out   long                    position );

  read_v      (inout array<ivec>             ivec     );
  write_v     (inout array<ivec>             ivec     );

  read_p      (in    string                  pattern,
               inout array<byte,1>           buffer,
               out   long                    len_out  );
  write_p     (in    string                  pattern,
               in    array<byte,1>           buffer,
               out   long                    len_out  );

  modes_e     (out   array<string,1>         emodes   );
  read_e      (in    string                  emode,
               in    string                  spec,
               inout array<byte,1>           buffer,
               out   long                    len_out  );
  write_e     (in    string                  emode,
               in    string                  spec,
               in    array<byte,1>           buffer,
               out   long                    len_out  );

  // Metric:
  //     name:  read
  //     desc:  fires if a file gets readable
  //     mode:  Read
  //     unit:  1
  //     type:  Bool
```

```
      //     value: True
      //
      //     name:  write
      //     desc:  fires if a file gets writable
      //     mode:  Read
      //     unit:  1
      //     type:  Bool
      //     value: "True"
      //
      //     name:  error
      //     desc:  fires if a file gets writable
      //     mode:  Read
      //     unit:  1
      //     type:  Bool
      //     value: "True"
    }
  }

+-------------------------------------------------------------+


Details:
========

  The current description covers the ubiquitous
  open/close/read/write/seek pattern, which is present
  in the vast majority of remote file access providers.

  class directory

    This class represents a directory containing physical
    files.

    - CONSTRUCTOR
      Purpose: open the directory
      Format:  CONSTRUCTOR        (in  string     url,
                                   in  int        flags,
                                   in  session    session,
                                   out directory  dir)
      Inputs:  url:                location of directory
               flags:              mode for opening
               session:            session to associate the
                                   object with
      Outputs: dir:                the newly created object
      Throws:  BadParameter
               DoesNotExist
      Notes:   - the session handle defaults to the SAGA
                 default session handle if not explicitely
                 specified
               - the default flag set is (None)
```

```
  - DESTRUCTOR
    Purpose: destroy the directory object
    Format:  DESTRUCTOR         (in  directory dir)
    Inputs:  dir:                the object to destroy
    Outputs: none


  Methods giving information about files:
  ---------------------------------------

  - get_size
    Purpose: returns the number of bytes in the file
    Format:  get_size           (in  string  name,
                                 in  int     flags,
                                 out long    size);
    Inputs:  name:               name of file to inspect
    Outputs: size:               number of bytes in the file
    Throws:  BadParameter
             DoesNotExist
    Notes:   - as 'st_size' field in the Unix call 'stat',
               as defined by POSIX

  - is_file
    Purpose: alias for is_entry in ns_directory


  Factory like methods for creating objects:
  ------------------------------------------

  - open_dir
    Purpose: creates a directory object
    Format:  open_dir           (in  string     name,
                                 in  int        flags,
                                 out directory  dir)
    Inputs:  name:               name of directory to open
             flags:              flags definition operation
                                 modus
    Outputs: dir:                opened directory instance
    Throws:  BadParameter
             DoesNotExist
    Notes:   - creates a new directory instance
             - currently there are no supported flags (FIXME)
             - default flag set is empty (None)
             - similar to opendir (3), as defined by POSIX


  - 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
      Throws:  BadParameter
               DoesNotExist
      Notes:   - if the file does not exist, it is created if
                 the 'Create' flag is given, otherwise it is
                 an error
               - it is an error if the file exists and both the
                 'Create' and the 'Excl' flag are given.
                 Otherwise the 'Excl' flag is ignored
               - the file is truncated to length 0 on the open
                 operation if the 'Trunc' flag is given
               - the file is in opened in append mode if the
                 'Append' flag is given (a seek (0, End) is
                 performed after the open)
               - 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 no respect to the given flags.
                 Note that a file can be opened in normal mode,
                 and then in locked mode, w/o an error getting
                 raised.  The lock will get removed on
                 destruction of the file object (that is on
                 close).  If an implementation does not
                 support locking, an descriptive error MUST
                 get issued if the 'Lock' flag is given.
               - default flag set is (Read).
               - similar to the open (2) call in Unix, as
                 specified by POSIX


  class file:

    This class represents an open file descriptor for
    read/write operations on a physical file.  It concept is
    similar to the file descriptor returned by the open (2)
    call in Unix.

    - CONSTRUCTOR
      Purpose: create the obj
      Format:  CONSTRUCTOR        (in  string        url,
                                   in  int           flags,
                                   in  session       session,
                                   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:  BadParameter
           DoesNotExist
  Notes:   - the session handle defaults to the SAGA
             default session handle if not explicitely
             specified
           - the default flag set is (Read)

- DESTRUCTOR
  Purpose: destroy the object
  Format:  DESTRUCTOR          (in  file             obj)
  Inputs:  obj:                 the object to destroy
  Outputs: none

- read
  Purpose: reads data from an open file
  Format:  read                (in  long   len_in,
                                in  string buffer,
                                out long   len_out);
  Inputs:  len_in:              length of data section to
                                be read
           buffer:              buffer to read into
  Outputs: len_out:             length of data read into
                                buffer
  Throws:  BadParameter
           ReadError
  Notes:   - reads up to len_in bytes from the file into
             the buffer.
           - 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.
           - the file pointer is positioned at the end of
             the byte area read during this call.
           - the given buffer must be large enough to
             store up to len_in bytes
           - similar to the read (2) call in Unix, as
             specified by POSIX

- write
  Purpose: write data into an open file
  Format:  write               (in  long   len_in,
                                in  string buffer,
                                out long   len_out);
  Inputs:  len_in:              number of bytes to be
                                written
           buffer:              bytes to be written
  Outputs: len_out:             number of bytes written
  Throws:  BadParameter
           WriteError
```

```
      Notes:   - writes up to len_in bytes from buffer into
                 the file at the current file position.
               - the file pointer is positioned at the end
                 of the byte area written during this call.
               - similar to the write (2) call in Unix, as
                 specified by POSIX

    - seek
      Purpose: reposition the file pointer
      Format:  seek               (in  long       offset,
                                   in  seek_mode  whence,
                                   out long       position);
      Inputs:  offset:             offset in bytes to move
                                   pointer
               whence:             offset is relative to
                                   'whence'
      Outputs: position:           position of pointer after
                                   seek
      Throws:  BadParameter
               WriteError
      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.
               - the file pointer can be positioned after the
                 end of the file w/o extending it.  Reads
                 behind EOF return Zeros.
               - similar to the lseek (2) call in Unix, as
                 specified by POSIX.

    - read_v
      Purpose: gather/scatter read
      Format:  read_v             (inout array<ivec> ivec);
      Inputs/
      Outputs: ivec:               array of ivec structs
                                   defining start (offset) and
                                   length (length) of each
                                   individual read, and buffer
                                   to read into, and integer to
                                   store result into.
      Throws:  BadParameter
               ReadError
      Notes:   - the behaviour of each individual read is as
                 in the normal read method.
               - the lengths returned also correspond to those
```

```
                of the normal read method.
              - an exception is thrown if any of the
                individual reads detects a condition which
                would raise an exception for the normal
                read method.
              - similar to the readv (2) call in Unix, as
                specified by POSIX

  - write_v
    Purpose: gather/scatter write
    Format:  write_v           (inout  array<ivec>   ivec);
    Inputs/
    Outputs: ivec:              array of ivec structs
                                defining start (offset) and
                                length (length) of each
                                individual write, and
                                buffers containing the data
                                to write
    Throws:  BadParameter
             WriteError
    Notes:   - the behaviour of each individual write is as
               in the normal write method.
             - the lengths returned also correspond to those
               of 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.
             - similar to the writev (2) call in Unix, as
               specified by POSIX


+-------------------------------------------------------------+


Examples:
=========

  Example: open a file. if its size is > 10, then read the
           first 10 bytes into a string, and print it.

   ---------------------------------------------------------------
   // c++ example
   void head (const char* url)
   {
     try {
       // get type and other infos
       saga::file my_file (url);

       off_t size = my_file.get_size ();
```

```
       if ( size > 10 )
       {
         char   buffer[11];
         long   bufflen;

         my_file.read (10, buffer, &bufflen);

         if ( bufflen == 10 )
         {
           printf ("head: '%s'\n", buffer);
         }
       }
     }

     // catch any possible error - see elsewhere for better
     // examples of error handling in SAGA
     catch ( const saga::exception & e )
     {
       std::cerr << "Oops! SAGA error: " + e.what () + std::endl;
     }

     return;
   }
   ----------------------------------------------------------------


+-----------------------------------------------------------------+



Notes:
======

  A 'stat' like method is not yet specified; the form of such an
  interface needs further consideration. However, the 'get_size'
  method provides the most frequent and well defined file size
  for now (the call may be deprecated when a stat specification
  is available).

  future API version may have something like:
  stat             (in  string          name,
                    out struct          statinfo );

+-----------------------------------------------------------------+
```

```
+--------------------------------------------------------------+

    #
    #         ####    ####      #     ####     ##    #
    #        #    #  #    #     #    #    #   #  #   #
    #        #    #  #          #    #       #    #  #
    #        #    #  #  ###     #    #       ######  #
    #        #    #  #    #     #    #    #  #    #  #
    #######   ####    ####      #     ####   #    #  ######

               #######
               #         #    #       ######
               #         #    #       #
               #####     #    #       #####
               #         #    #       #
               #         #    #       #
               #         #    ######  ######

+--------------------------------------------------------------+


Summary:
========

   There are a number of replica catalogue systems implemented or
   in development.  This API is the intersection of features
   common to these implementations.  (TODO: enumerate these
   systems.)

   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 namespace package with operations for operating
   on entries in replica catalogues.

+--------------------------------------------------------------+


Specification:
==============

  package saga.logical_file
  {
    enum open_dir_flags
    {
      /* Placeholder */
    }

    enum open_flags
    {
```

```
    Unknown    =  -1,
    None       =   0,
    Create     =   1,
    Excl       =   2,
    Lock       =   4,
    Truncate   =   8,
    Append     =  16,
    Read       =  32,
    Write      =  64,
    ReadWrite  = 128
  }


  class logical_directory : implements-all saga::object
                            implements-all saga::attribute
                            implements-all saga::monitorable
                            extends        saga::ns_directory
  {

    CONSTRUCTOR     (in  string                  url,
                     in  array<open_dir_flags,1> flags,
                     in  session                 session,
                     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  array<open_dir_flags,1> flags,
                     out logical_directory       dir);

    open            (in  string                  name,
                     in  array<open_flags,1>     flags,
                     out logical_file            file);
  }


  class logical_file : implements-all saga::object
                       implements-all saga::attribute
                       implements-all saga::monitorable
                       extends        saga::ns_entry
  {
    CONSTRUCTOR     (in  string                  url,
                     in  array<open_flags,1>     flags,
                     in  session                 session,
                     out logical_file            file);
    DESTRUCTOR      (in  logical_file            file);
```

```
      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,1>         names);
      replicate       (in string                   name);
    }
  }

+-------------------------------------------------------------+


Details:
========

  class logical_directory

    This class represents a container for logical files in a
    logical file catalog.  It allows traversal of the catalogs
    name space, and the manipulation and creation (open) of
    logical files in that name space.

    Factory like methods for creating objects (see note in
    saga::name_space specification)

    - CONSTRUCTOR
      Purpose: create the object
      Format:  CONSTRUCTOR      (in  string        url,
                                 in  array<open_dir_flags,1>
                                                   flags,
                                 in  session       session,
                                 out logical_directory
                                                   obj)
      Inputs:  url:              location of directory
               flags:            mode for opening
               session:          session to associate with
                                 the object
      Outputs: obj:              the newly created object
      Throws:  BadParameter
               DoesNotExist
      Notes:   - the session handle defaults to the SAGA
                 default session handle if not explicitely
                 specified
               - currently there are no supported flags
               - the default flag set is (None)
               - similar to opendir (3), as defined by POSIX

    - DESTRUCTOR
      Purpose: destroy the object
```

```
      Format:  DESTRUCTOR         (in  logical_directory obj)
      Inputs:  obj:                the object to destroy
      Outputs: none


    - is_file
      Purpose: alias for is_entry in ns_directory

    - open_dir
      Purpose: creates a new logical_directory instance
      Format:  open_dir         (in  string           name,
                                 in  array<open_dir_flags,1>
                                                       flags,
                                 out logical_directory dir);
      Inputs:  name:             name of directory to open
               flags:            flags definition operation
                                 modus
      Outputs: dir:              opened directory instance
      Throws:  BadParameter
               DoesNotExist
      Notes:   - notes to logical_directory constructor apply


    - open
      Purpose: creates a new logical_file instance
      Format:  open             (in  string         name,
                                 in  array<open_flags,1>
                                                     flags,
                                 out logical_file    file);
      Inputs:  name:             file to be opened
               flags:            flags definition operation
                                 modus
      Outputs: file:             opened file instance
      Throws:  BadParameter
               DoesNotExist
      Notes:   - notes to logical_file constructor apply



  class logical_file:

    This class provides means to handle the contents of Logical
    Files.

    That contents consists of strings representing locations
    of physical files associated with the logical file.  In
    general, these locations could be logical files as well.
    In fact, they are usually handled as opaque strings, and
    no assumption about validity or the nature of the target
    of the location is made.  Exception: see the replicate
    and add_location method description.
```

```
- CONSTRUCTOR
  Purpose: create the object
  Format:  CONSTRUCTOR      (in  string        url,
                             in  array<open_dir_flags,1>
                                                flags,
                             in session         session,
                             out logical_file   obj)
  Inputs:  url:              location of directory
           flags:            mode for opening
           session:          session to associate with
                             the object
  Outputs: obj:              the newly created object
  Throws:  BadParameter
           DoesNotExist
  Notes:   - the session handle defaults to the SAGA
             default session handle if not explicitely
             specified
           - if the file does not exist, it is created if
             the 'Create' flag is given, otherwise it is
             an error
           - it is an error if the file exists and both the
             'Create' and the 'Excl' flag are given.
             Otherwise the 'Excl' flag is ignored
           - the file is truncated to length 0 on the open
             operation if the 'Trunc' flag is given.  For
             logical files that means: no physical file
             location is associated with the logical file.
           - the file is in opened in append mode if the
             'Append' flag is given.  For logical files
             that means: newly added physical file
             locations are appended to the set of known
             locations.
           - 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 no respect to the given flags.
             Note that a file can be opened in normal mode,
             and then in locked mode, w/o an error getting
             raised.  The lock will get removed on
             destruction of the file object (that is on
             close).  If an implementation does not support
             locking, an descriptive error MUST get issued
             if the 'Lock' flag is given.
           - default flag set is (Append)
             Note that the default is different to the
             class saga::file.
           - similar to the open (2) call in Unix, as
             specified by POSIX
```

```
- DESTRUCTOR
  Purpose: destroy the object
  Format:  DESTRUCTOR       (in  logical_file   obj)
  Inputs:  obj:              the object to destroy
  Outputs: none


- add_location
  Purpose: add a name to the location set
  Format:  add_location     (in string name);
  Inputs:  name:             location to add to set
  Outputs: none
  Throws:  BadParameter
           AlreadyExists
  Notes:   - this methods adds a given string  to the set
             of locations associated with the logical file.
           - if the location is already in the set, no
             error is issued.
           - the implementation may choose to interpret the
             locations associated with the logical file
             instance.  It may return an error indicating
             an invalid location if it is unable or
             unwilling to handle that specific location.
           - the documentation MUST specify how valid
             location are contructed.


- remove_location
  Purpose: remove a name from location set
  Format:  remove_location  (in string name);
  Inputs:  name:             location to remove from set
  Outputs: none
  Throws:  BadParameter
           DoesNotExist
  Notes:   - this method removes a given string from the
             set of locations associated with the logical
             file.
           - if the location is not in the set of
             locations, an error is issued.
           - if the set of locations is empty after that
             operation, the logical file object is still a
             valid object (see replicate methods
             description).


- update_location
  Purpose: change a name in location set
  Format:  update_location  (in string name_old,
```

```
                              in string name_new);
    Inputs:  name_old         location to be updated
             name_new         update for location
    Outputs: none
    Throws:  BadParameter
             DoesNotExist
    Notes:   - this method removes a given string from the
               set of locations associated with the logical
               file, and adds a new string.
             - if the old location is not in the set of
               locations, an error is issued.


  - list_locations
    Purpose: list the locations in the location set
    Format:  list_locations   (out array<string,1> names);
    Inputs:  none
    Outputs: names:            array of locations in set
    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 - see
               description to the remove_location method.

  - replicate
    Purpose: replicate a file from any of the known
             locations to a new location, and add the new
             location to the location set on success.
    Format:  replicate        (in string name);
    Inputs:  name:             location to replicate to
    Outputs: none
    Throws:  BadParameter
    Notes:   - the method requests a two step operation:
               1) copy an entity at any of the locations
                  associated with the logical file to the
                  given string, which represents a new
                  location.
               2) perform an add_location for the given
                  string.
             - the method is not required to be atomic, but:
             - the method is required to be either
               successfull in both steps, or to issue an
               error indicating if both methods failed, or if
               only one of the methods succeeded (leaving the
               system in an inconsistent state).
             - a replicate call on an instance with empty
               location set results in an error.
             - this methods requires the implementation of
             the class to interpret the locations
               associated with the logical file instance.  If
               that is impossible, an error indicating an
```

```
                  invalid location must be issued.


+-------------------------------------------------------------+


Examples:
=========

  // c++ example
  int main ()
  {
    saga::logical_file lf ("lfn://remote.catalog.net/tmp/file1");

    lf.replicate ("gsiftp://localhost.net/tmp/file.rep");

    saga::file f ("gsiftp://localhost.net/tmp/file.rep");
    std::cout << "sice of local replica: "
              << f.get_size ()
              << std::endl;
  }

+-------------------------------------------------------------+


Notes:
======

  It is recommended to interpret the locations associated with
  logical files with valid locations for saga::file, and to have
  the implementation using saga::file.  That helps to program
  coherently with the saga::name_space, saga::file and
  saga::logical_file packages.

  logical_file and logical_directory should implement the
  saga::attribute interface.

  logical_directory should implement a find method searching on
  saga::attribute.

+-------------------------------------------------------------+
```

```
+--------------------------------------------------------------+

                    #####
                        #   ####   #####
                        #  #    #  #    #
                        #  #    #  #####
                   #    #  #    #  #    #
                   #    #  #    #  #    #
                    #####    ####   #####

+--------------------------------------------------------------+

Summary:
========

  Many of the use cases provided to the SAGA-RG had either
  explicit or implied requirements for submitting jobs to grid
  resources, and for monitoring and controlling these submitted
  jobs.

  This API provides an interface for submitting jobs to a grid
  resource, either in batch mode, or in an interactive mode. It
  also provides APIs for controlling these submitted jobs (e.g.
  to terminate, suspend, or signal a running job), and APIs for
  retrieving status information for both running and completed
  jobs.

  The goals of this API are to provide enough functionality to
  satisfy the requirements of grid developers according to the
  "80-20" rule.  This API is also intended to incorporate the
  work of the DRMAA-WG, and to extend the API based on the
  experience of implementing DRMAA.  Much of this specification
  was taken directly from DRMAA, with many of the differences
  arising from an attempt to make the job API consistent with the
  overall SAGA API model. Note [1].

+--------------------------------------------------------------+

Specification:
==============

  package saga.resource
  {
    // enum job_state = saga::state (from task package)

    class job_definition : implements-all saga::object
                           implements-all saga::attribute
    {
      CONSTRUCTOR (out job_definition obj);
      DESTRUCTOR  (in  job_definition obj);
```

```
    }

    class job : implements-all saga::object
                implements-all saga::attribute
                implements-all saga::monitorable
                extends        saga::task
    {
      /* no CONSTRUCTOR */
      DESTRUCTOR           (in  job             job);

      // job inspection
      get_job_id           (out string          job_id);
      get_state            (out state           state);
      get_state_detail     (out string          detail);
      get_job_definition   (out job_definition  job_def);
      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_definition   job_def);
      signal               (in int              signum);
      terminate            (void);
    }

    class job_service : implements-all saga::object
                        implements-all saga::monitorable
    {
      CONSTRUCTOR          (in  session         session,
                            out job_service     service);
      DESTRUCTOR           (in  job_service     service);

      create_job           (in  string          rm,
                            in  job_definition  job_def,
                            out job             job);
      run_job              (in  string          rm,
                            in  string          commandline,
                            out opaque          stdin,
                            out opaque          stdout,
                            out opaque          stderr,
                            out job             job);
      list                 (out array<string,1> job_ids);
      get_job              (in  string          job_id,
                            out job             job);
      get_self             (out job             job);
    }
  }
```

```
+-------------------------------------------------------------+

Details:
========

  class job_definition:
  ---------------------

    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. The only required attribute in order to
    perform a valid job submission is the 'JobCmd'. Given the
    'JobCmd', 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).

    At the end of the description of an attribute there is a bit
    in parentheses that indicates whether a particular attribute
    is supported within a particular system. Tokens include
    DRMAA, JSDL, LSF, OpenPBS, PBSPro, SGE and Condor, and are
    intended to be extended by members of the working group.

    The attributes encapsulated within this class are:

    'JobCmd'
    - The command to execute. This is the only required
      attribute.  Can be a full pathname, or a pathname
      relative to the 'JobCwd' as evaluated on the execution
      host. String.  (DRMAA, JSDL, LSF)

    'JobArgs'
    - Positional parameters for the command. Vector of strings.
      (DRMAA, JSDL, LSF)

    'JobState'
    - The job state at submission. jobs can be submitted into a
      suspend or hold state such that they need manual resume
      before being considered for scheduling. Valid values are
      "Hold", "Suspend".  If not specified, the job will enter the
      default "Pending" state. Type String.  (DRMAA, LSF)
```

```
    'JobEnv'
    - The set of environment variables which will be exported
      to the environment of the started job. The string format
      is "name=value". Vector of strings. (DRMAA, JSDL)

    'JobCwd'
    - The working directory for the job. If this is a relative
      path, it will be treated as relative to the users home
      directory on the system where the job runs. String.
      (DRMAA, JSDL)

    'JobInteractive'
    - Run the job in interactive mode. This means that stdio
      streams will stay connected to the submitter after job
      submission, and during job execution. The stdio streams
      are retrieved by calling the getXStream methods of the
      jobs class.  Boolean. (LSF)

    'JobStdin'
    - The pathname of the standard input file. If this is a
      relative pathname, it will be treated as relative to the
      users home directory on the system where the job runs.
      String. (DRMAA, JSDL, LSF)

    'JobStdout'
    - The pathname of the standard output file. If this is a
      relative pathname, it will be treated as relative to the
      users home directory on the system where the job runs.
      String. (DRMAA, JSDL, LSF)

    'JobStderr'
    - The pathname of the standard error file. If this is a
      relative pathname, it will be treated as relative to the
      users home directory on the system where the job runs.
      String. (DRMAA, JSDL, LSF)

    'JobContact'
    - A set of endpoints describing where to report job
      completion status, as well as other resource manager
      defined state transitions. The format of the string will
      be that of a URI (e.g. fax:+123456789, sms:+123456789,
      mailto:csmith@platform.com). Vector of strings. (DRMAA
      (email addresses), LSF (email addresses))

    'JobNotification'
    - A flag which indicates whether to send notifications to
      endpoints listed in 'JobContact'. Mostly used to shut
      off notifications if they are on by default. Boolean.
      (DRMAA, LSF)

    'JobName'
```

```
  - The job name to be attached to the job submission.
    String.  (DRMAA, LSF)

  'JobNative'
  - The native specification as described in the DRMAA
    specification. Note [3]. This value is passed as is to
    the backend without any meaning or semantics within the
    SAGA API. String. (DRMAA)

  'FileTransfer'
  - A list of file transfer directives which can be used to
    transfer files to the execution host of the job before
    the job is run, and to transfer files from the execution
    host of the job when the job completes. Vector of
    strings. (DRMAA (limited), JSDL (much enhanced), LSF)

    The syntax of a file transfer directive 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.

  'JobStartTime'
  - The time after which a job is considered for scheduling.
    Could be viewed as a desired job start time, but that is
    up to the resource manager. Date/time. (DRMAA, LSF)

  'Deadline'
  - Specifies a hard deadline after which the resource
    manager should terminate the job. Date/time. (DRMAA, LSF)
```

```
    'WallclockHardLimit'
    - Specifies a hard limit on the amount of wall clock time
      in seconds that a job may consume, after which the
      resource manager should terminate the job. Integer.
      (DRMAA, JSDL, LSF)

    'WallclockSoftLimit'
    - Provides an estimate of the amount of wall clock time in
      seconds which a job will require. This attribute is
      intended to provide hints to the scheduler. If this time
      limit is reached, the action taken is specific to the
      resource manager and its scheduling policies. Integer.
      (DRMAA, LSF)

    'Cputime'
    - Estimated job runtime in CPU seconds. The CPU time is
      aggregated across all processes/threads of the job.
      Integer.  (LSF)

    'NumCpus'
    - The total number of cpus requested for this job. How the
      cpus are allocated is determined by the policy of the
      resource manager, and can possibly be affected by the
      'Native' attribute if the resource manager supports
      it. Integer. (JSDL, LSF)

    'Memory'
    - Estimated maximum amount of memory that the job requires
      in Megabytes. The memory usage of the job is aggregated
      across all processes of the job. Float. (JSDL, LSF)

    'ProcessorType'
    - Select compatible processor for job submission. The list
      of allowed values is taken from the JSDL specification
      jsdl:ProcessorArchitectureEnumeration. Note [4]. String.
      (JSDL)

    'OperatingSystem'
    - Select compatible operating system for job submission.
      The list of allowed values is taken from the JSDL
      specifications jsdl:OperatingSystemTypeEnumeration. Note
      [4]. String. (JSDL)

    'HostList'
    - A list of host names, or host group names, which can be
      considered by the resource manager as candidate hosts for
      the job. Whether or not the job actually ends up running
      on one of the hosts in the list, is solely at the
      discretion of the resource manager. Vector of strings.
      (JSDL, LSF)
```

```
    'Queue'
    - The name of a queue to place the job into. While SAGA
      itself does not define the semantics of "queue", many
      back end systems can make use of this attribute. String.
      (LSF)


    - CONSTRUCTOR
      Purpose: create the object
      Format:  CONSTRUCTOR        (out job_definition obj)
      Inputs:  none
      Outputs: obj:                the newly created object
      Notes:   - a job_definition is NOT associated to a
                 session, but can be used for job services
                 from different sessions.


    - DESTRUCTOR
      Purpose: destroy the object
      Format:  DESTRUCTOR         (in  job_definition obj)
      Inputs:  obj:                the object to destroy
      Outputs: none


  class job:
  ----------

    The job provides the manageability interface to a job
    submitted to a resource manager. There are two general types
    of methods: those for retrieving job state and information,
    and those for manipulating the submitted job. The methods
    intended to manipulate jobs cannot make any guarantees about
    how the resource manager will effect an action to be taken.
    Please see note [5].

    job implements the 'Attribute' interface, and understands
    the following attribute names.  If not noted otherwise, none
    of these attributes is available before the job is running,
    and none is guaranteed to have a value if the job is running
    or after the job finishes.  Also, the attributes can change
    their values during the lifetime of the job.

    'ExecutionHosts'
    - The list of host names or IP addresses which were
      allocated to run this job. Vector of strings.

    'Created'
    - The time stamp of the job creation in the resource
      manager (i.e. the submission time). Date/time.

    'Started'
```

```
    - The time stamp indicating when the job started running.
      Date/time.

    'Finished'
    - The time stamp indicating when the job completed.
      Date/time.

    'Cputime'
    - The number of cpu seconds consumed by the job. The value
      is aggregated across all processes/threads of the job.
      Integer.

    'MemoryUse'
    - The current aggregate memory usage in megabytes of the
      processes of this job, or the memory high water mark when
      the job is complete. Integer.

    'VmemoryUse'
    - The current aggregate virtual memory usage in megabytes
      of the processes of this job, or the virtual memory high
      water mark when the job is complete. Integer.

    'JobCwd'
    - The current working directory of the job on the execution
      host(s). This can be used to determine the location of
      files that are staged using relative file paths. String.

    'ExitCode'
    - The process exit code as collected by the wait(2) series
      of system calls. The exit code is collected from the
      process which was started from the 'JobCmd' attribute
      of the jobDefinition object.  Is only available in final
      states, but not guaranteed.  Integer.

    'Signaled'
    - Indicates whether the job exited due to receipt of a
      signal.  Is only available in final states, but not
      guaranteed.  Boolean.

    'Termsig'
    - The signal number which caused the job to exit.  Is only
      available in final states, but not guaranteed.  Integer.


    - CONSTRUCTOR
      This class has no constructor, and can only be obtained by
      calling job_server.create_job() or job_server.run_job().


    - DESTRUCTOR
      Purpose: destroy the object
```

```
    Format:  DESTRUCTOR         (in  job  obj)
    Inputs:  obj:                the object to destroy
    Outputs: none


  - get_job_id
    Purpose: Get the resource managers representation of
             the job identifier.
    Format:  get_job_id         (out string job_id);
    Inputs:  none
    Outputs: job_id:                job identifier string


  - get_state_detail
    Purpose: Retrieve details of the current job state.
    Format:  get_state_detail (out string detail);
    Inputs:  none:
    Outputs: deatail:              details about the current
                                   job state
    Notes:   - The SAGA job state model is a very simplified
               model.  The job state details obtained with
               this calls returns information from any more
               explicit state model of the back end.
             - BES compliant states SHOULD be returned
             - the format of the state details SHOULD be:
               "<model>:<state>", with valid models being:
               "BES", "DRMAA", "GRAM", or implementation
               specified. The state should be spelled
               like "UpperCase" (example: "BES:StagingIn").


  - get_job_definition
    Purpose: Retrieve the job_definition which was used to
             submit this job instance.
    Format:  get_job_definition (out job_definition job_def);
    Inputs:  none
    Outputs: job_def:               a job_definition object
    Notes:   - There are cases when the job_definition is not
               available, and thus this object will be null.
               These include cases when the job might not have
               been submitted through SAGA, and get_job() was
               used to retrieve a job, or this state
               information has been lost (e.g. the client
               application restarts and the particular SAGA
               implementation did not persist the information).


  - get_stdin
    Purpose: retrieve input stream for a job.
    Format:  get_stdin          (out opaque stdin)
    Inputs:  none
```

```
    Outputs: stdin:              standard input stream for
                                 the job
    Notes:   - If the job was submitted as interactive (the
             'JobInteractive' attribute was set at job
             submission), this method retrieves the standard
             input stream for the job. The type of the
             stream is indicated in SIDL as opaque, since
             this type will be rendered differently based
             on the language bindings, and will be made
             concrete in another specification document
             which describes language bindings.


  - get_stdout
    Purpose: retrieve output stream of job
    Format:  get_stdout         (out opaque stdout)
    Inputs:  none
    Outputs: stdout:             standard output stream for
                                 the job
    Notes:   - If the job was submitted as interactive (the
               'JobInteractive' attribute was set at job
               submission), this method retrieves the
               standard output stream for the job.


  - get_stderr
    Purpose: retrieve error stream of job
    Format:  get_stderr         (out opaque stderr)
    Inputs:  none
    Outputs: stderr:             standard error stream for
                                 the job
    Notes:   - If the job was submitted as interactive (the
               'JobInteractive' attribute was set at job
               submission), this method retrieves the
               standard error stream for the job.


  - suspend
    Purpose: Ask the resource manager to perform a suspend
             operation on the running job.
    Format:  suspend ();
    Inputs:  none
    Outputs: none
    Notes:   - The semantics of suspend, and the action taken
               to suspend a job is resource manager specific.


  - resume
    Purpose: Ask the resource manager to perform a resume
             operation on the running job.
    Format:  resume ();
```

```
    Inputs:  none
    Outputs: none
    Notes:   - The semantics of resume, and the action taken
               to resume a job is resource manager specific.


  - checkpoint
    Purpose: Ask th resource manager to initiate a checkpoint
             operation on a running job.
    Format:  checkpoint ();
    Inputs:  none
    Outputs: none
    Notes:   - The semantics of checkpoint, and the actions
               taken to initiate a checkpoint, are resource
               manager specific.


  - migrate
    Purpose: Ask the resource manager to migrate a running
             job to another host.
    Format:  migrate              (in job_definition job_def);
    Inputs:  job_def:              new job parameters to apply
                                   when the job is migrated
    Outputs: none
    Notes:   - The call may also be used to change some
               parameters of a non-finished job (e.g. change
               runtime limit estimates, etc). The action of
               migration might change the job identifier within
               the resource manager.
             - job_def might indicate new resource requirements,
               for example.


  - terminate
    Purpose: Ask th resource manager to terminate a dispatched
             job
    Format:  terminate ();
    Inputs:  none
    Outputs: none
    Notes:   - the job can be in in Running or Suspended state
             - the semantics of terminate, or the action taken,
               is specific to the resource manager.


  - signal
    Purpose: Ask th resource manager to deliver an arbitrary
             signal to a dispatched job.
    Format:  signal               (in int signum);
    Inputs:  signum:               signal number to be
                                   delivered
    Outputs: none
```

```
      Notes:   - The semantics of signal, or the action taken,
                 is specific to the resource manager. There is no
                 guarantee that the signal number specified is
                 valid for the operating system on the execution
                 host where the job is running.


  class job_service:
  ------------------

    The job_service provides an class for job creation and
    discovery.

    - CONSTRUCTOR
      Purpose: create the object
      Format:  CONSTRUCTOR        (in  session       session,
                                   out job_service   obj)
      Inputs:  session:            session to associate with
                                   the object
      Outputs: obj:                the newly created object
      Notes:   - the session handle defaults to the SAGA
                 default session handle if not explicitely
                 specified


    - DESTRUCTOR
      Purpose: destroy the object
      Format:  DESTRUCTOR         (in  job_service   obj)
      Inputs:  obj:                the object to destroy
      Outputs: none



    - create_job
      Purpose: create a job instance
      Format:  create_job         (in  string         rm,
                                   in  job_definition job_def,
                                   out job            job);
      Inputs:  rm:                 rm name or IP address of
                                   the resource manager which
                                   will accept and run the job
               job_def:            description of job to be
                                   submitted
      Outputs: job:                a job object representing
                                   the submitted job instance
      Throws:  IncorrectParameter
      Notes:   - the returned job is in the New state
               - calling run() on the job will submit it to
                 the resource, and advance its State.
```

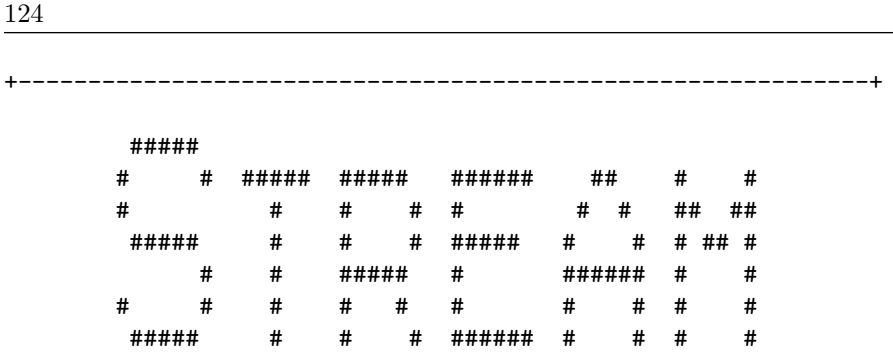
```
- run_job
  Purpose: Run a command synchronously.
  Format:  run_job            (in  string  rm,
                               in  string  commandline,
                               out opaque  stdin,
                               out opaque  stdout,
                               out opaque  stderr,
                               out job     job);
  Inputs:  rm:                 rm name or IP address of
                               the resource manager which
                               will accept and run the job
           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
  Notes:   - This is a convenience routine built on the
             create_job method, and is intended to simplify
             the steps of creating a job_definition,
             creating and running the job, and then
             querying the stdio streams.


- list
  Purpose: Get a list o jobs which are currently known by
           the resource manager.
  Format:  list               (out array<string,1> job_ids);
  Inputs:  none
  Outputs: job_ids:            an array of job identifiers
  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.


- 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
```

```
    - get_self
      Purpose: This method returns a job object representing
               _this_ job, i.e. the calling application.
      Format:  get_self             (out job    job)
      Inputs:  none
      Outputs: job:                  a job object representing
                                     _this_ job.


  job_id:
  -------

    The job ID is treaded as opaque string in the SAGA API.
    However, for reasons of compatibility and potential extended
    use of the job id information, the job id SHOULD be
    implemented as:

      "[backend url]-[native id]"

    For example, a job submitted via ssh, and having the unix pid
    1234, should get the job id:

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

+-------------------------------------------------------------+

Examples:
=========

  Example : simple job submission and polling for finish.

  // ------------------------------------------------
  // c++ example
  std::list <char*>    transfers;
  saga::job_definition jobdef;

  transfers.push_back ("infile > infile");
  transfers.push_back ("ftp://host.net/path/out << outfile");

  jobdef.set_attribute        ("'JobCmd'",  "myjob.sh");
  jobdef.set_attribute        ("'NumCpus'", "16");
  jobdef.set_vector_attribute ("'FileTransfer'", transfers);

  saga::job_service myjs;
  saga::job         myjob = myjs.create_job ("remote.host.net", jobdef);

  myjob.run ();

  while ( 1 )
  {
```

```
    saga::state state = myjob.get_state ();

    if ( saga::job::Running == state.saga () )
    {
      std::list <char*> hostlist = myjob.get_attribute
                                   ("'ExecutionHosts'");
      // print hostlist
    }
    else if ( saga::job::Done == state.saga () )
    {
      print "Job completed successfully.";
      exit;
    }
    else
    {
      // saga state can be Unknown or Failed

      char* exitcode = myjob.get_attribute ("'ExitCode'");

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

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

```
+-------------------------------------------------------------+

Notes:
======


  References:
  -----------

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

  [2] The job_state enumerated type encapsulates the possible
      states of a job. They are copied from the BES WG as of
      February 2006.
      https://forge.gridforum.org/projects/ogsa-bes-wg/document/ogsa-bes-draft-v16/en/1

  [3] http://www.ggf.org/documents/GWD-R/GFD-R.022.pdf

  [4] https://forge.gridforum.org/projects/jsdl-wg

  [5] 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.

  [6] In attributes that take paths and pathnames, there was
      some discussion as to whether we should require the
      implementation of placeholders which could represent things
      like 'home directory', and that are not known until the job
      is bound to an execution host.

  [7] There is discussion as to which interfaces might be
      missing. One possibility was a job history retrieval
      interface could be necessary. This could be used to map
      state transitions of a job throughout its lifetime.

  [8] The DRMAA 'job category' attribute was left out of the
      strawman API. During the discussions of this attribute
      within the design team meetings, it was deemed to simplify
      the API at the expense of the implementor of the back end
      system. Thus, it was left out pending discussion.

+-------------------------------------------------------------+
```



Summary:
========

A number of use cases involved launching of remotely located components in order to create distributed applications. The use cases require simple remote socket connections to be established between these components and their control interfaces.

The target of this streams API is to establish the simplest possible authenticated socket connections with hooks to support authorization and encryption schemes. The API is

1) Not performance oriented: If you need performance, then it is better to program directly to the APIs of existing performance oriented protocols like GridFTP or XIO.

2) 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 from being supported).

3) Does not attempt to create a programming paradigm that diverges very far from baseline BSD sockets, Winsock, or Java Sockets interfaces.

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 suitable for applications that do not have too sophisticated requirements (as per 80-20 rule). As applications become more sophisticated, they can graduate to more sophisticated native APIs in order to support those needs.

```
Specification:
==============

  package saga.stream
  {
    enum stream_state
    {
      Unknown      = -1,
      Error        =  1,
      Open         =  2,
      Dropped      =  3,
      NotConnected =  4
    }

    enum activity_type
    {
      Unknown      = -1,
      Read         =  1,
      Write        =  2,
      Exception    =  4,
      Any          =  7,
    }

    class stream : implements-all saga::object
                   implements-all saga::attribute
                   implements-all saga::monitorable
    {
      CONSTRUCTOR  (in    string           url,
                    in    session          session,
                    out   stream           obj);
      DESTRUCTOR   (in    stream           obj);

      get_url      (out   string           url);
      get_context  (out   context          info);

      connect      (void);
      state        (out   stream_state     state);
      wait         (in    activity_type    what,
                    in    double           timeout,
                    out   array<activity_type,1>
                                           activity);

      read         (inout array<byte,1>    buffer,
                    in    long             buffer_size,
                    out   long             bytes_read);
      write        (in    array<byte,1>    buffer,
                    in    long             size,
                    out   long             bytes_written);
      close        (void);
```

```
// Attributes:
//   name:  bufsize
//   desc:  determines the size of the send buffer, in bytes
//   mode:  ReadWrite
//   type:  Int
//   value: <system dependend>
//
//   name:  timeout
//   desc:  determines the amount of idle time
//          before dropping the line, in seconds
//   mode:  ReadWrite
//   type:  Int
//   value: <system dependend>
//
//   name:  blocking
//   desc:  determines if read/writes are blocking
//          or not
//   mode:  ReadWrite
//   type:  Bool
//   value: True
//
//   name:  compression
//   desc:  determines if data are compressed
//          before/after transfer
//   mode:  ReadWrite
//   type:  Bool
//   value: <system dependend>
//
//   name:  nodelay
//   desc:  determines if packets are sent
//          immediatley, i.e. w/o delay
//   mode:  ReadWrite
//   type:  Bool
//   value: True
//
//   name:  reliable
//   desc:  determines if all sent data MUST arrive
//   mode:  ReadWrite
//   type:  Bool
//   value: True


// Metrics:
//   name:  read
//   desc:  fires if a stream gets readable
//   mode:  Read
//   unit:  1
//   type:  Bool
//   value: True
//
//   name:  write
```

```
      //   desc:  fires if a stream gets writable
      //   mode:  Read
      //   unit:  1
      //   type:  Bool
      //   value: True
      //
      //   name:  exception
      //   desc:  fires if a stream has an error condition
      //   mode:  Read
      //   unit:  1
      //   type:  Bool
      //   value: True
      //
      //   name:  any_event
      //   desc:  fires if the stream gets readable,
      //          writable, or has an error condition
      //   mode:  Read
      //   unit:  1
      //   type:  Bool
      //   value: True
      //
      //   name:  dropped
      //   desc:  fires if the stream gets dropped by the
      //          remote party
      //   mode:  Read
      //   unit:  1
      //   type:  Bool
      //   value: True
      //
      //   name:  state
      //   desc:  fires if the state of the stream changes,
      //          and and has the value of the stream state
      //          enum
      //   mode:  Read
      //   unit:  1
      //   type:  enum
      //   value: Unknown
    }


    class stream_server : implements-all saga::object
                          implements-all saga::attributes
                          implements-all saga::monitorable
    {
      CONSTRUCTOR       (in    string          url,
                         in    session         session,
                         out   stream_server    obj);
      DESTRUCTOR        (in    stream_server    obj);

      get_url            (out   string          url);
```

```
      wait              (in    double          timeout,
                         out   stream          stream);

      // Metrics:
      //   name:  client_connect
      //   desc:  fires if a client connects
      //   mode:  Read
      //   unit:  1
      //   type:  Bool
      //   value: True
    }
  }

+-------------------------------------------------------------+


Details:
========

  class stream:

    This is the object that encapsulates all client stream
    objects.

    - CONSTRUCTOR
      Purpose: Constructor, initializes a client client stream,
               for later connection to an server.
      Format:  CONSTRUCTOR          (in  string  url,
                                     in  session session,
                                     out stream  stream);
      Inputs:  url:                  server location in URL
                                     syntax
               ctx:                  SAGA context used for
                                     stream setup
      Outputs: stream:               new, unconnected stream
                                     instance
      Notes:   - the session handle defaults to the SAGA
                 default session handle if not explicitely
                 specified
               - returned stream is NULL on error.
               - server location and possibly protocol is
                 described by the input URL.
               - a saga::context is necessary to authenticate
                 the socket.
               - The socket is only connected after the connect
                 method is called in order to support two-phase
                 connections that appear in some authentication
                 schemes.  The state of the socket upon
                 construction is therefore NotConnected.  Once
                 the connect() method is sucessfully called, the
                 state will change to Open.
```

```
- DESTRUCTOR
  Purpose: Destructor, closes any active connection and
           deallocates any memory consumed by the stream
           data structures.
  Format:  DESTRUCTOR           (in stream stream)
  Inputs:  stream:               stream data structure that
                                 needs to be closed and
                                 deallocated.
  Outputs:  none
  Notes:   - Because the data structures might consume some
             memory space internally, even closed, dropped,
             or failed sockets must be deallocated using
             the destroystream method.

- close
  Purpose: closes an active connection
  Format:  close                (void)
  Inputs:  none
  Outputs: none
  Throws:  IncorrectState
  Notes:   - IncorrectState is thrown when the stream was
             closed previously

- connect
  Purpose: Establishes a connection to the target defined
           during the construction of the stream.
  Format:  connect              ();
  Inputs:  none
  Outputs: none
  Throws:  IncorrectState
  Notes:   - on success, the streams state is changed to
             Open


- read
  Purpose: Read a raw buffer from socket.
  Format:  read                 (inout string        buffer,
                                 in    long          size,
                                 out   long          nbytes);
  Inputs:  buffer:              Empty buffer passed in to
                                get filled
           size:                Maximum number of bytes
                                that can be copied in to
                                the buffer.
  Outputs: nbytes:              number of bytes read, if
                                successful. (0 is also
                                valid)
  Throws:  IncorrectState
  Notes:   - This call is blocking.  Use wait or poll
             methods to implement non-blocking reads.
```

```
- write
  Purpose: Write a raw buffer to socket.
  Format:  write               (in  string        buffer,
                                in  long          size,
                                out long          nbytes);
  Inputs:  buffer:              raw array containing data
                                that will be sent out via
                                socket
           size:                number of bytes of data in
                                the buffer
  Outputs: nbytes:              bytes written if successful
  Throws:  IncorrectState
  Notes:   - This call is blocking.  Use wait method to
             implement polling for non-blocking writes.

- state
  Purpose: Check on the state of an active connection.
  Format:  state               (out stream_state state);
  Inputs:  none
  Outputs: state:               state of stream
  Notes:   - the only valid states for a stream are:
             Error:
               The socket has entered a non-fatal error
               state.  If the state is fatal, then the
               state will be Dropped.  The reason for
               the error must be queried through a separate
               interface (not yet defined).
             NotConnected:
               This the state for a newly created socket
               where the connect method has not been
               invoked.
             Open:
               This is the state for an active/connected
               socket.
             Dropped:
               This is the state for a socket where the
               remote side of the socket connection has been
               lost or some other error has broken the
               connection.  A socket will enter the dropped
               state if authentication fails for example.
               The actual reason for the drop must be
               queried through the error handling interface.
           - this method is only returning the *state* of the
             stream and not the reason it entered that state.
           - more states can be added as required

- wait
  Purpose: Allows the stream to be interrogated to find out
           if it is ready for reading/writing, or if it has
           entered an error state.
```

```
    Format:  wait                 (in  activity_type what,
                                   in  double       timeout,
                                   out <array,1>activity_type
                                                    cause);
    Inputs:  what:                 parameter list of activity
                                   types to wait for
             timeout:              number of seconds to wait
    Outputs: cause:                activity type causing the
                                   call to return
    Throws:  IncorrectState
    Notes:   - wait will only check on the conditions specified
               in the 'what' parameter list (a bitmask in some
               language bindings).  The options are
                 Read:
                   The socket has pending data available for
                   reading.
                 Write:
                   The socket is available for writing.
                 Exception:
                   If the socket has entered an error state or
                   the remote host has dropped the connection.
                 Any:
                   This is shorthand for any of the above
             - the call returns enum describing the
               availability of the socket (eg. readable,
               writable, or exception) masked against the
               input 'what' enum list.
             - the call is blocking if the timeout is any
               positive value.  It blocks forever (no timeout)
               if the timeout value is < 0.0.  The wait method
               can be used for polling if the timeout is set to
               zero.  The wait method will only check for the
               activity_type that is specified in the call (and
               ignore all other issues).


  - get_context
    Purpose: Gets a security context object from an OPEN
             (connected)
    Format:  get_context          (out context      context);
    Inputs:  none
    Outputs: context:              a context object.
    Throws:  IncorrectState
    Notes:   - throws IncorrectState exception or <null>
               context object returned if the security
               info is inapplicable (non-authenticated sockets)
             - the context returned contains the security
               information from the REMOTE party, and can be used
               for authorization.
             - It is assumed that the context is authenticated.
```

```
  - get_url
    Purpose: get URL used for creating the string
    Format:  get_url               (out string url);
    Inputs:  none
    Outputs: url:                  string containing the URL
                                   of the connection.
    Thorws:  nothing
    Notes:   - this is the URL which can be passed to
               stream constructor to create another
               connection to the same stream_server.



class stream_server:

  The stream_server object establishes a listening/server object
  that waits for client connections.  It can *only* be used as
  a factory for Server sockets.  It doesnt do any read/write
  I/O.

  - CONSTRUCTOR
    Purpose: Constructor, to create a new stream_server object
    Format:  CONSTRUCTOR          (in  string       url,
                                   in  session      session,
                                   out stream_server stream);
    Inputs:  url:                  channel name or url,
                                   defines the source side
                                   binding for the stream
                                   (eg. the port number for
                                   the service)
             session:              session to be used for
                                   object creation
    Outputs: stream:               new stream_server object
    Notes:   - the session handle defaults to the SAGA
               default session handle if not explicitely
               specified
             - returns NULL stream_server object on error
             - the context is primarily used to hide the
               security information necessary to establish
               authenticated connections.

  - DESTRUCTOR
    Purpose: Destructor for stream_server object.
    Format:  DESTRUCTOR           (in stream_server stream)
    Inputs:  stream:               stream_server object to be
                                   destroyed
    Outputs: none
    Notes:   - the call cleans up any memory used by the
               stream_server object in addition to closing the
```

```
                service port.

    - wait
      Purpose: wait for incoming client connections
      Format:  wait                (in  double timeout,
                                    out stream client);
      Inputs:  timeout:             number of seconds to wait
                                    for client
      Outputs: client:              new Connected stream object
      Throws:  IncorrectState
      Notes:   - supports either blocking or polling for new
                 client connections.
               - if successful, it returns a new stream object
                 that is connected to the client.
               - unlike new client streams, the new connection is
                 return in the Connected state.
               - returns NULL or equivalent if it times out.
               - returns NULL or equivalent if connection setup
                 failed
               - timeout < 0.0 wait forever
               - timeout > 0.0 wait this number of seconds
               - timeout = 0.0 poll and return immediately.

    - get_url
      Purpose: get URL to be used to connect to serverstream
      Format:  get_url             (out string url);
      Inputs:  none
      Outputs: url:                 string containing the URL
                                    of the connection.
      Thorws:  nothing
      Notes:   - this is the URL which can be passed to
                 stream constructor to create a connection to
                 this stream_server.


+-------------------------------------------------------------+


Examples:
=========

  Sample SSL/Secure Client:
  -------------------------

    Opens a stream connection using native security: context is
    passed in implicitly via a global SAGA context
    (GSI or SSL security)

    // C++/JAVA Style
       int recvlen;
       saga::stream s ("localhost:5000");
```

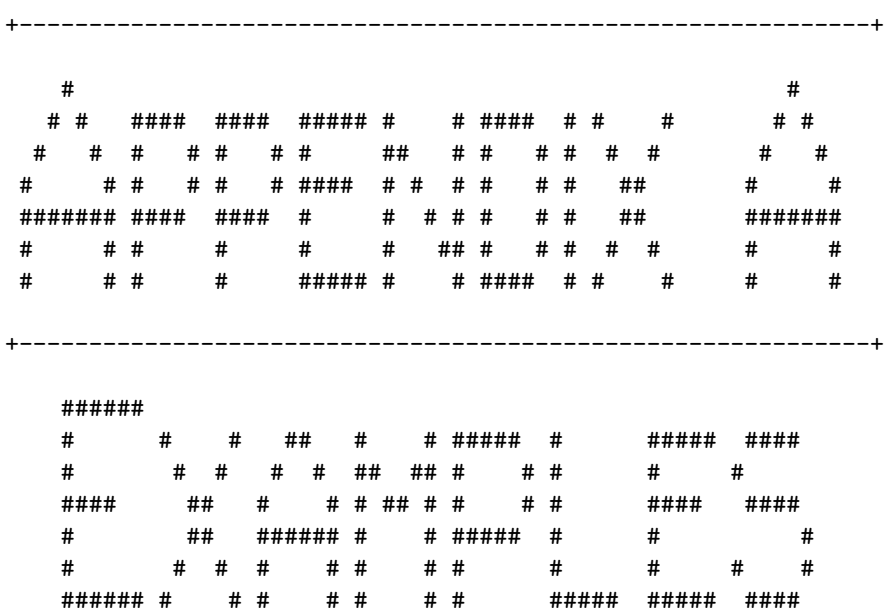
```
      s.connect ();
      s.write   ("Hello World!", 12);

      // blocking read, read up to 128 bytes
      recvlen = s.read (buffer, 128);


   /* C Style */
      int recvlen;

      SAGA_stream = SAGA_Stream_open ("localhost:5000");

      SAGA_Stream_connect (s);
      SAGA_Stream_write   (s, "Hello World!", 12);

      /* blocking read, read up to 128 bytes */
      recvlen = SAGA_Stream_read (s, buffer, 128);


    c  Fortran Style */
       INTEGER   err,SAGAStrRead,SAGAStrWrite,err
       INTEGER*8 SAGAStrOpen,streamhandle
       CHARACTER buffer(128)
       SAGAStrOpen("localhost:5000",streamhandle)
       call SAGAStrConnect(streamhandle)
       err = SAGAStrWrite(streamhandle,"localhost:5000",12)
       err = SAGAStrRead(streamhandle,buffer,128)


  Sample Secure Server:
  ---------------------

    Once a connection is made, the server can use information
    about the authenticated client to make an authorization
    decision

     // C++/JAVA Style
        saga::stream_server server ("tcp://localhost/5000");
        saga::stream        client;
        int                 done = 0;

        // now wait for a connection (normally in a loop)
        do {
          string value;
          // wait forever for connection
          client = server.wait ();

          // get the distinguished name (DN)
          saga::context = client.get_context ();
```

```
          // check if context type is X509, and if DN is the
          // authorized one
          if ( saga::context::X509 == context.type () &&
               ! strcmp (context.get_attribute ("DN"), auth_dn) )
          {
            done = 1; // allowed
          }
          else
          {
            SAGA::stream_close (client); // not allowed
          }
        } while ( ! done );

        // start activity on client socket...


  Example for async stream server
  -------------------------------

    // c++ example
    class my_cb : public saga::callback
    {
      privat:
        saga::stream_server ss;
        saga::stream        s;

      public:

        my_cb (saga::stream_server ss_,
               saga::stream        s)
        {
          ss = ss_;
          s   = s_;
        }

       ~my_cb (void) { }

        void callback (saga::monitorable mt,
                       saga::metric      m,
                       int               c)
        {
          s = ss_.wait ();
          mt.remove_callback (c);  // want to be called only once
        }
     }

     int main ()
     {
       saga::stream_server ss;
       saga::stream        s;
       my_cb cb (ss, s);
```

```
      ss.add_callback ("client_connect", cb);

      while ( true )
      {
        if ( s.state != saga::stream::Open )
        {
          // no client, yet
          sleep (1);
        }
        else
        {
          // handle open socket
          s.write ("Hello Client\r\n", 14);
          s.close ();

          // restart listening
          ss.add_callback ("client_connect", cb);
        }
      }

      return (-1); // unreachable
    }
```

```
+-------------------------------------------------------------+
```

```
Notes:
======

+-------------------------------------------------------------+
```



Introduction
============

This appendix shows a couple of API examples in different languages. As stated in the global introduction, these examples are not supposed to be normative -- language bindings are outside the scope of this document. This appendix is rather supposed to illustrate how the authors imagine the use of the API in various languages.

We hope that the examples illustrate, that the API stays SIMPLE in various language incarnations, as was the major design intent for the _S_AGA API.

Example 1: Files:
=================

open a file. if its size is > 10, then read the first 10 bytes into a string, print it, end return it.

Example 1a: C++

```
// c++ example
void head (const char* url)
```

```
{
  try {
    // get type and other infos
    saga::file my_file (url);

    off_t size = my_file.get_size ();

    if ( size > 10 )
    {
      char   buffer[11];
      long   bufflen;

      my_file.read (10, buffer, &bufflen);

      if ( bufflen == 10 )
      {
        printf ("head: '%s'\n", buffer);
      }
    }
  }

  // catch any possible error - see elsewhere for better
  // examples of error handling in SAGA
  catch ( const saga::exception & e )
  {
    std::cerr << "Oops! SAGA error: " + e.what () + std::endl;
  }

  return;
}
-----------------------------------------------------------------
-----------------------------------------------------------------
Example 1b: C
-------------
  char* head (const char* url)
  {
    SAGA_File my_file = SAGA_File_create (url);

    if ( NULL == my_file )
    {
      fprintf (stderr, "Could not create SAGA_File for %s: %s\n",
               url, SAGA_Session_get_error (theSession));
      return (NULL);
    }

    off_t size = SAGA_File_get_size (my_file);

    if ( size < 0 )
    {
      fprintf (stderr, "Could not determine file size for %s: %s\n",
               url, SAGA_Session_get_error (theSession));
```

```
      return (NULL);
    }
    else if ( size > 10 )
    {
      char   buffer[11];
      size_t bufflen;

      ssize_t ret = SAGA_File_read (my_file, 10, buffer, &bufflen);

      if ( ret < 0 )
      {
        fprintf (stderr, "Could not read file %s: %s\n",
                 url, SAGA_Session_get_error (theSession));
        return  (NULL);
      }

      if ( bufflen == 10 )
      {
        buffer [11] = '\0';
        printf ("head: '%s'\n", buffer);
        return (buffer);
      }
      else
      {
        fprintf (stderr, "head: read is short! %d\n", bufflen);
        return  (NULL);
      }
    }

    fprintf (stdout, "head: file is too small! %d\n", size);

    return  (NULL);
  }

----------------------------------------------------------------
Example 1c: Java
----------------

import saga*;

class MyClass
{
  // open a file. if its size is > 10, then read the first
  // 10 bytes into a string, print it, end return it.
  string head (URI uri)
  {
    try
    {
      saga::file f (uri);

      if ( 10 <= f.get_size () )
```

```
        {
          FileInputStream in (uri);
          byte[]          buffer = new buffer[10];
          int             res    = in.read (buffer);

          if ( 10 == res )
          {
            System.out.println ("head: " + buffer);
          }
          else
          {
            System.err.println ("head: read is short! " + res);
          }

          return new string (buffer);
        }
        else
        {
          System.out.println ("file is too small: " + size);
        }
      }

      // catch any possible error - see elsewhere for better
      // examples of error handling in SAGA
      catch (...)
      {
        System.out.println ("Oops!");
      }

      return null;
    }
  }


  ----------------------------------------------------------------
  Example 1d: Perl ('normal' error handling)
  ----------------------------------------

    sub head ($)
    {
      my $url     = shift;
      my $my_file = new saga::file (url)
                or die ("can't create file for $url: $!\n");

      my $size    = my_file->get_size ();

      if ( size > 10 )
      {
        my $buffer = my_file->read (10)
                or die ("can't read from file $url: $!\n");
```

```
        if ( length ($buffer == 10 ) )
        {
          print "head: '$buffer'\n";
          return ($buffer);
        }
        else
        {
          printf "head: short read! %s\n", saga::get_error ();
        }
      }
      else
      {
        print "file $url is too short: $size\n";
      }

      return (undef);
    }

  ------------------------------------------------------------
  Example 1e: Perl (exceptions)
  -----------------------------

    sub head ($$)
    {
      my $session = shift;
      my $url     = shift;

      eval {
        my $my_file = new saga::file (session, url);

        my $size    = my_file->get_size ();

        if ( size > 10 )
        {
          my $buffer = my_file->read (10);

          if ( length ($buffer == 10 ) )
          {
            print "head: '$buffer'\n";
            return ($buffer);
          }
          else
          {
            printf "head: short read! %s\n", saga::get_error ();
          }
        }
        else
        {
          print "file $url is too short: $size\n";
        }
      }
```

```
      if ( $@ =~ /saga/i )
      {
        print "catched saga error: $@\n" if $@;
      }

      return (undef);
    }

  ------------------------------------------------------------
  Example 1f: Fortran
  -------------------

    TBD

  ------------------------------------------------------------
  Example 1f: Python
  ------------------

    TBD

+-------------------------------------------------------------+
```