GWD-R
Category: Recommendations

Manuel Pereira – IBM Almaden Research Center
Osamu Tatebe - Grid Technology Research Center
Leo Luan, IBM Almaden Research Center
Ted Anderson, IBM Almaden Research Center
Jane Xu, IBM Systems and Technology Group

GFS-WG

November 2004

Resource Namespace Service Specification

Status of This Memo

This memo provides information to the Grid community about file system directory services. It does not define any standards or technical recommendations. Distribution is unlimited.

Abstract

This document describes the specification of a Resource Namespace Service (RNS), which is a WSRF compliant web service capable of providing namespace services for any entity interested in registering an End Point Reference or URL with an easily accessible, hierarchically managed, name. This service, previously referred to as a virtual filesystem directory service (VFDS), has been updated to incorporate an interface design that utilizes document style messages as described in the WSRF specification. RNS is intended to facilitate namespaces services for a wide variety of Grid services, with an initial emphasis as one of the essential services for Grid file systems or virtual file systems in the Grid environment. It can be employed to manage the namespace of federated and virtualized data, services, or effectively any resource capable of being referenced in a grid/web environment. This document proposes a set of operations and resource property definitions that define the Resource Namespace Service.

Contents

mpereira@us.ibm.com

o.tatebe@aist.go.jp

Manuel Pereira – IBM Almaden Research Center
Osamu Tatebe - Grid Technology Research Center
Leo Luan, IBM Almaden Research Center
Ted Anderson, IBM Almaden Research Center
Jane Xu, IBM Systems and Technology Group

mpereira@us.ibm.com
o.tatebe@aist.go.jp

Introduction

Data in the Grid can be of any format and be stored in any type of storage system. There can be many hundreds of petabytes of data in grids, among which a very large percentage is stored in files. A standard mechanism to describe and organize file-based data is essential for facilitating access to this large amount of data. The Grid File System Working Group (GFS-WG) was established in GGF data area to standardize a mechanism to address this need by providing a Grid File System (GFS) or virtual file system in the Grid environment.

Two major deliverables of the WG are (1) architecture of Grid File System Services and (2) specification of namespace services. This document is intended to address (2) by proposing a set of WSRF-compliant operations exploiting well-defined and abstract resource properties that define the service. The Resource Namespace Service (RNS) is fully capable of providing namespace services for any web or grid-related resource, however for the specific purpose of addressing the deliverables for the GFS-WG this document will focus on the application of RNS providing namespace services for Grid file systems. In this capacity, RNS will manage the namespace of federated and virtualized data, access control mechanisms, and meta-data management [1]. It will provide features such as (a) virtualized hierarchical namespaces for data resources (such as files, filesystems, live data feeds, database queries, etc.), (b) the facilitation of efficient and transparent file sharing, and (c) the ability to describe and manage file-system and application-specific metadata.

The overall architecture of the Grid File System will be specified later in GFS-WG, which provides infrastructure of virtual file systems facilitating federation and sharing of virtualized data from file systems in the Grid environment by using Resource Namespace Services.

Resource Namespace Services

The Resource Namespace Service, which will henceforth be referred to as RNS, enables construction of a uniform, global, hierarchical namespace.[1] This directory service or namespace service enables federation of essentially any web or grid resource. RNS embodies a three-tier naming architecture, which consists of a *human-readable*, *logical* or *abstract*, and *resource reference* names. Name-to-resource mapping in RNS features the optional arrangement of two levels of indirection. The first level of indirection is realized by mapping human-readable names to direct resource references. Since the address properties of the direct resource reference may be modified without altering the RNS entries that refer to them, this simple approach offers a convenient means of name-to-resource mapping with a single level of indirection. A second level of indirection may be appreciated when mapping human-readable names to logical names, which in turn map logical names to direct resource references and hence the second level of indirection. The advantage of using a logical name is that logical names may be referenced and resolved using RNS independent of the hierarchical namespace. This means that logical names may be used as a globally unique logical resource identifier and be referenced directly by both the RNS namespace as well as other services; note that mapping information and associated pointer handles for direct resource references are not exposed by RNS and are therefore used exclusively by RNS. Following is a diagram that illustrates the three-tier naming architecture; please note that this diagram does not employ abstract (non-data) references, but rather is intended to illustrate the levels of the naming architecture:

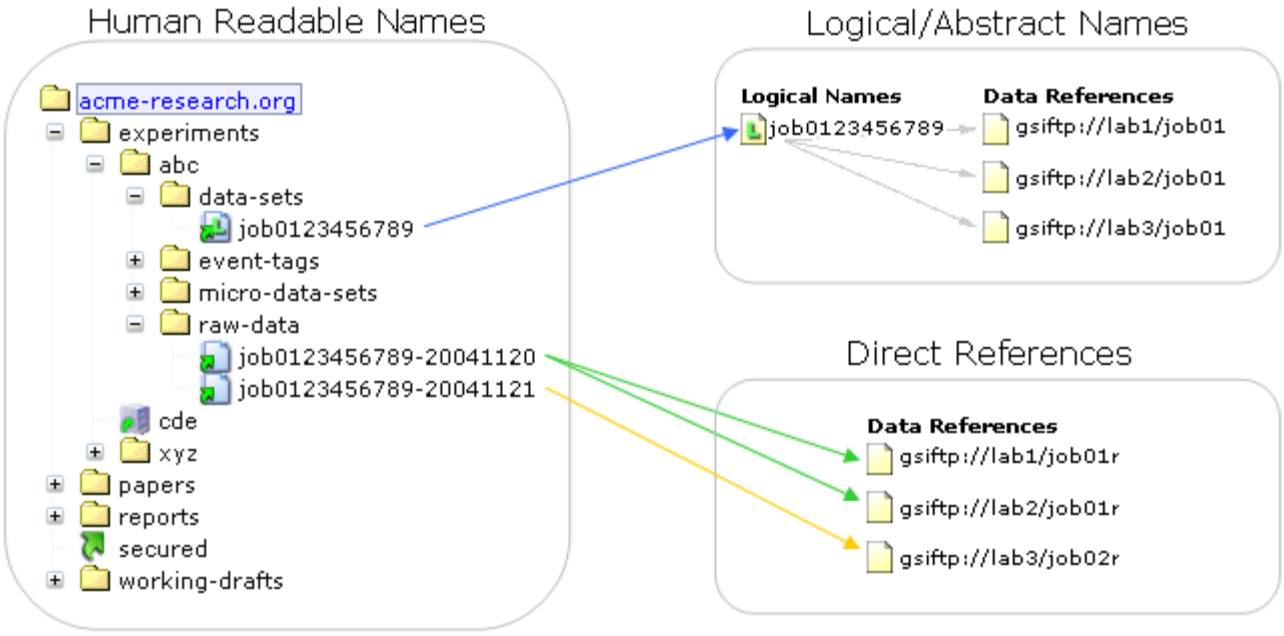

Figure 1 - Three-Tier Naming Architecture

1.1 Basic Namespace Components

RNS is comprised of two fundamental namespace components: *virtual directories* and *junctions*. These two essential namespace components, also referred to as RNS *entries*, are employed to federate existing resources and construct a uniform hierarchy. A description of each follows:

A virtual directory is an RNS entry that is represented as a non-leaf node in the hierarchical namespace tree. When rendered by a filesystem client, a virtual directory appears as a standard filesystem directory, however does not have any corresponding position in any physical filesystem; hence it is *virtual*. A virtual directory, therefore, is purely a namespace entity that functions in much the same way as a conventional filesystem directory by maintaining a list of subentries, which thereby demonstrate a hierarchical relationship. There are no restrictions regarding the layout of the namespace tree; both virtual directories and junctions can be nested within nested virtual directories recursively.

A junction is an RNS entry that interconnects a reference to an existing resource into the global namespace. It functions in much the same way as a traditional distributed file system mount point with the unique property of maintaining uniform namespace representation while facilitating two levels of indirection. Junctions are categorized into five basic types: *abstract resource junctions*, *data resource junctions*, *logical junctions*, *referrals*, and *aliases*. An *abstract resource junction* is an entry that maps to at least one web or grid resource by way of an WS-Addressing[3] End Point Reference (EPR) or URL. This is a many-to-many mapping, meaning that one entry may reference many resources and one resource may be referenced by many entries. There is no limitation as to what may be referenced by RNS provided that a WS-Addressing compliant EPR, or an RFC 1738 compliant URL, is used to register the reference mapping. A *data resource junction* points to at least one data reference, thereby maintaining a many-to-many mapping between junction and data reference (data references are described in greater detail in the following section). A *logical junction* is a junction that either contains an EPR/URL that points to a secondary service, like a

Replica Location Service (RLS), for logical-to-physical resolution given a logical name, or it contains an internal reference to a logical data reference entry within RNS. *Referrals* are junctions that link to other RNS instances, thereby facilitating such features as federation of independent domains of control, scalability of a single domain of control, availability of redundant service instances that may or may not be geographically distributed, etc; an example referral is illustrated in Figure 1 as "secured", its URL might look something like: rns://rns.secured.abc.com/. An *alias* entry is a junction that references another entry within the same service instance for hard-links, and by EPR/URL including path for soft-links/symbolic-links; the behaviour described here is comparable to conventional Unix symbolic-links and hard-links. In all cases, junctions are capable of maintaining a list of references (EPRs/URLs) per entry, that is a single junction my render several available EPRs, each of which represent replicas or copies of the same data or service instance.

When dealing with data, RNS enables federation of individual files as well as filesystem trees that are exported by a variety of storage systems. An example of a file reference is illustrated by "job0123456789-20041120" and an example of a filesystem reference is alluded to by "cde", both in Figure 1.

The following sections explore the objects and interface definitions that exemplify the operations of RNS. This material is not comprehensive, is subject to change, and does not examine the internal procedures of the interface.

1.2 WSRF Document Style Interface

RNS employs a document style message-based service interface that implements most of the WS-ResourceProperties[4] document types. The previous object oriented model has been subsumed by a stateful exchange of SOAP messages. With the implementation of the WS-Resource specification, RNS offers stateful interaction by maintaining a stateful resource referred to as a context. To begin stateful use of RNS a client sends a message requesting to establish a new context and proceeds to leverage the *implied resource pattern*[4] for maintaining a "current working directory" marker in subsequent message exchanges. This is particularly useful when traversing deep hierarchies since any previously rendered portion of the namespace tree will not need to be traversed again.

In addition to implementing a stateful resource for maintaining state between client and service, RNS implements the GetMultipleResourceProperties message exchange for all query oriented operations and the SetMultipleResourceProperties message exchange for all change oriented operations. Access to RNS entry metadata is therefore achieved by using a resource properties request document that indicates which properties to retrieve. This means that only the properties the client is interested in are retrieved. Furthermore, when submitting a change request message to the service, only the properties specified will be SOAP encoded and sent to the service. As a result, a greater efficiency, with respect to the sheer size of the SOAP message, may be realized.

The RNS port type (RNSPortType) extends the GetResourceProperty port type defined by WS-ResourceProperties[3], implementing the getResourceProperty, getMultipleResourceProperties, and setResourceProperty operations.

1.2.1 Resource Properties Documents

RNS specifies two primary resource properties documents that together exemplify the details of the service. These resource properties documents correspond to the two foundational service objects that construct the information presented in the namespace service, the *Entry* and *DataReference* objects. The resource properties document design is extensible in that properties can be added

without requiring modification of the core service. This denotes that there are no defined methods for access member data (properties), in fact the namespace objects that such methods could operate on are not defined in the specification. We only refer to them as "objects" in a conceptual manner, understanding that they are not classes that will be instantiated in the client runtime environment. Just as runtime objects may contain a number of data members, RNS exports a number of properties that are associated with each namespace entry and data reference entry from the datastore on the server side. Therefore, performing a standard message exchange (operation) to retrieve information about a particular namespace entry is initiated by a message request containing a list of all of the property names (QName) who's values are to be retrieved, and completes by returning a SOAP message containing the values of all of the properties requested. The returned values may contain nested value arrays and therefore are properly decoded by traversing the entire SOAP message, which is comprise of nest-able message elements.

Following are the currently available properties for the Entry and Data Reference objects, for Context (stateful resource), along with a list of message element names that may be used in message exchanges:

Entry Resource Properties

QName	SOAP Type
AliasCount	xsd:int
ChildCount	xsd:int
Description	xsd:string
ModificationTime	xsd:dateTime
Name	xsd:string
Alias	xsd:boolean
File	xsd:boolean
Filesystem	xsd:boolean
Hardlink	xsd:boolean
Logical	xsd:boolean
Referral	xsd:boolean
VirtualDirectory	xsd:boolean
AbstractJunction	xsd:boolean

Data Reference Resource Properties

QName	SOAP Type
Checksum	xsd:string
ChecksumType	xsd:string
Complete	xsd:boolean
MutableSource	xsd:boolean
ReadOnly	xsd:boolean
ReplicaCopy	xsd:boolean
Size	xsd:long
Target	xsd:string
Timestamp	xsd:dateTime
Version	xsd:string

Context Resource Properties

QName	SOAP Type
Path	xsd:string

Message Elements

QName
AbstractReference
AbstractReferences
DataReference
DataReferences
AllEntryProperties
AllDataRefProperties
Entry
RNS
RNSKey

1.2.2 Property Relationships

Since RNS is SOAP 1.1 compliant and allows for message exchanges between heterogeneous runtime environments, it does not enforce appropriate property relationships, dependencies, or exclusivities. The service will however enforce such relationship requirements on the server side, but a good understanding of what correct property relationships are is helpful.

Notice that the entry resource includes a number of boolean properties. Among the boolean properties, the *VirutalDirectory* property is mutually exclusive, which is to say that if it is “true” then all of the other boolean values MUST be “false”. Combinations of “true” and “false” values are potentially valid from the remaining properties if *VirutalDirectory* is “false”. An exhaustive list of will be given in a later revision, for now just know that a valid relationship must exist between the properties specified and their respective values.

Property relationship is not an issue for data reference resources, their boolean properties are not mutually exclusive.

1.3 Operations of the Resource Namespace Service

RNS is composed of the following types of operations:

1) Operations for querying namespace and data reference information.
2) Operations for creating, removing, moving/renaming, and updating entries and data references.
3) Operations for managing attributes or status of an entry or data reference.

1.3.1 Operation Parameters

Please note that in the current WSRF implementation by Globus 3.9.3, only one parameter is permitted per operation. Before examining the purposed operations, it is necessary to review the associated operation parameters.

1.3.1.1 QueryInput

This is a document literal service compliant object (complexType) that contains five elements:

```
<xsd:complexType name="QueryInput">
  <xsd:sequence>
    <!-- Absolute or relative path, or NULL for current directory -->
    <xsd:element ref="tns:path"          minOccurs="1" maxOccurs="1"/>
    <!-- Starting subentry index for list operations -->
    <xsd:element ref="tns:index"         minOccurs="1" maxOccurs="1"/>
    <!-- Number of subentries to return per list operation message -->
    <xsd:element ref="tns:count"         minOccurs="1" maxOccurs="1"/>
    <!-- If true, resolve all logical and data references per entry -->
    <xsd:element ref="tns:resolve"       minOccurs="1" maxOccurs="1"/>
    <!-- Array of QNames used to indicate what properties to retrieve -->
    <xsd:element ref="tns:propertyTypes" minOccurs="1" maxOccurs="unbound"/>
  </xsd:sequence>
</xsd:complexType>
```

1.3.1.2 ChangeInput

This is a document literal service compliant object (complexType) that contains three elements:

```
<xsd:complexType name="ChangeInput">
  <xsd:sequence>
    <!-- Absolute or relative path, or NULL for current directory -->
    <xsd:element ref="tns:path"          minOccurs="1" maxOccurs="1"/>
    <!-- Used only move operation to indicate the new destination -->
    <xsd:element ref="tns:newPath"       minOccurs="1" maxOccurs="1"/>
    <!-- WS-ResourceProperties SetResourceProperties -->
    <xsd:element name="changeProperties" ref="wsrp:SetResourceProperties"
                                         minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

1.3.1.3 DataReferenceInput

This is a document literal service compliant object (complexType) that contains three elements:

```
<xsd:complexType name="DataReferenceInput">
  <xsd:sequence>
    <!-- Logical or abstract name -->
    <xsd:element ref="tns:logicalName"   minOccurs="1" maxOccurs="1"/>
    <!-- Direct Resource Reference name -->
    <xsd:element ref="tns:physicalName"  minOccurs="1" maxOccurs="1"/>
    <!-- WS-ResourceProperties SetResourceProperties -->
    <xsd:element name="changeProperties" ref="wsrp:SetResourceProperties"
                                         minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

1.3.2 Namespace Operations

More detailed documentation for operations will be included in future revisions of this specification.

1.3.2.1 create

Enables a client to submit a request message that contains an array of message elements, each of which represent a property name/value pair, to be created and persistently stored on the server. This operation is primarily used for the creation of namespace entries, but may also effect the creation of other datastore objects (like End Point Reference entries if the service implementation utilizes a separate entry for storing EPR information).

Parameter: *ChangeInput*

path:	The absolute or relative path of the parent virtual directory where this entry should be created as a subentry.
changeProperties:	The MessageElement array.

Returns: *SetMultipleResourcePropertiesResponse*

1.3.2.2 delete

Enables a client to submit a request message that contains the path of the entry to delete.

Parameter: *path*: The absolute or relative path of the entry to be deleted.

Returns: *SetMultipleResourcePropertiesResponse*

1.3.2.3 list

Enables a client to submit a request message that contains an array of property names to be retrieved for each namespace entry that is a subentry of the virtual directory entry denoted by the path value within the input parameter.

Parameter: *QueryInput*

path:	The absolute or relative path of the parent virtual directory to list.
index:	The starting subentry index; allows for retrieving segments of the subentry list.
count:	Number of subentries to return per list message exchange.
resolve:	If true, resolve all logical references and data references per entry.
propertyTypes:	Array of QNames used to indicate what properties to retrieve,

Returns: *GetMultipleResourcePropertiesResponse*

1.3.2.4 lookup

Enables a client to submit a request message that contains an array of property names to be retrieved for the namespace entry denoted by the path value within the input parameter.

Parameter: *QueryInput*

path: The absolute or relative path of the parent virtual directory to list.
resolve: If true, resolve all logical references and data references per entry.
propertyTypes: Array of QNames used to indicate what properties to retrieve,

Returns: *GetMultipleResourcePropertiesResponse*

1.3.2.5 move

Enables a client to submit a request message that request a namespace entry be moved or renamed.

Parameter: *ChangeInput*

path:	The absolute or relative path of the original entry to be moved.
newPath:	The absolute or relative path of the destination entry.

Returns: *SetMultipleResourcePropertiesResponse*

1.3.2.6 update

Enables a client to submit a request message that contains an array of message elements, each of which represent a property name/value pair, to be used to update an existing entry in the database.

Parameter: *ChangeInput*

path:	The absolute or relative path of the entry to be updated.
changeProperties:	The MessageElement array.

Returns: *SetMultipleResourcePropertiesResponse*

1.3.3 Data Reference Operations

More detailed documentation for operations will be included in future revisions of this specification.

1.3.3.1 createDataReference

Enables a client to submit a request message that contains an array of message elements, each of which represent a property name/value pair, to be created and persistently stored on the server.

Parameter: *DataReferenceInput*

logicalName:	The name of the data reference to be created; zero length if the data reference to be created is supposed to be a physical data reference.
physicalName:	The name of the data reference to be created; zero length if the data reference to be created is supposed to be a logical data reference.
changeProperties:	The MessageElement array.

Returns: *SetMultipleResourcePropertiesResponse*

1.3.3.2 deleteDataReference

Enables a client to submit a request message that contains the necessary name of the data reference to delete.

Parameter: *DataReferenceInput*

logicalName:	The name of the data reference to be deleted; zero length if the data reference to be deleted is a physical data reference.
physicalName:	The name of the data reference to be deleted; zero length if the data reference to be deleted is a logical data reference.

Returns: *SetMultipleResourcePropertiesResponse*

1.3.3.3 mapLogicalDataReference

Enables a client to submit a request message that contains an array of message elements, each of which represent a property name/value pair, to be used to create/update a mapping between a logical data reference and a physical data reference in the database.

Parameter: *DataReferenceInput*

logicalName:	The name of the logical data reference to be mapped.
physicalName:	The name of the physical data reference being mapped.
changeProperties:	The MessageElement array.

Returns: *SetMultipleResourcePropertiesResponse*

1.3.3.4 resolve

Enables a client to submit a request message that contains the logical name of a logical data reference to be resolved.

Parameter: *logicalName:* The name of the logical data reference to be resolved.

Returns: *GetMultipleResourcePropertiesResponse*

1.3.3.5 updateDataReference

Enables a client to submit a request message that contains an array of message elements, each of which represent a property name/value pair, to be used to update an existing data reference in the database.

Parameter: *DataReferenceInput*

logicalName:	The name of the data reference to be updated; zero length if the data reference to be updated is a physical data reference.
physicalName:	The name of the data reference to be updated; zero length if the data reference to be updated is a logical data reference.
changeProperties:	The MessageElement array.

Returns: *SetMultipleResourcePropertiesResponse*

Federation of Resource Namespace Services

A global namespace service directly implies the employment of a multitude of namespace servers by virtue of geographical distribution, segregated domains of ownership and control, scalability, and redundancy/availability. A principal goal of a global namespace service is to provide a location independent view of consistent access paths to resources. Since these access paths are represented by hierarchal path names, symbolizing a globally unique identifier to a given resource, it is a natural extension of the design to postulate an architecture that federates multiple namespace servers in a hierarchical fashion. Similar to the well established DNS model, RNS servers can be interlinked by referrals whilst providing a seamless and transparent view of the namespace. Once several instances of the namespace service are interlinked, the most obvious challenge is related to path name resolution when dealing with paths that cross referral boundaries. There are two fundamental approaches to resolving path names that span multiple namespace domains or service instances: service referrals and delegated resolution.

1.4 Service Referrals

The most straightforward and arguably the most secure and truly scalable approach to resolving path names that span multiple domains or service instances is to place the onus of handling RNS referrals on the RNS client. In this approach, the namespace server would simply return a RNS referral to the RNS client when a junction to another namespace server is encountered. The client implementing the RNS API is then responsible for continuing the task of resolving the original path name by connecting to the namespace server indicated by the RNS referral and querying the newly connected server for further (relative) path name resolution.

One clear advantage of this approach is the direct management of namespace service connections, which implies authentication and authorization control per connection, rather than accessing a referred namespace server via proxied security. Additionally, this approach promotes distributed work load balancing; instead of requiring RNS servers to handle namespace requests for both locally managed namespace and remotely managed namespace via proxy.

1.5 Delegated Resolution

Another possible approach to resolving path names that span multiple domains or service instances is to empower the RNS server to delegate queries to other RNS servers for complete resolution of any given path. Although this approach is demonstrated in DNS, it should be noted that the security requirements are quire different. Since DNS generally operates in a public read-only manner without authentication and authorization per DNS server, it is not too unreasonable to endorse such an approach. RNS, however, facilitates the possibility of requiring authentication per service instance and enforcing access control per entry. Nevertheless, an approach that allows for the possibility of delegated resolution should be considered as at least an optional mode of operation; incidentally DNS is capable of both approaches.

Considerations

There are several issues to consider, with respect to RNS, which have not been explored in this document.

- Security – The topic of security as a whole is not discussed in this specification document. Security is recognized as a substantial are of interest and will require further investigation.
- Replication of RNS databases – To enhance fault tolerance and reliability, replication of namespace service data is indispensable. The consistency model required by RNS needs to be investigated.
- Backup – Backup of RNS data may be required.
- Discussion of access control lists (ACLs) within RNS, their purpose, scope, representation, and enforcement. If access permissions defined by physical filesystems are to be represented within RNS then significant consideration must be taken with respect to consistency problems between access permissions of a virtual file and the corresponding file data.
- Removal or modification of a file data without notification to the file system directory services.
- Consistency problems between file data replicas.
- Interoperability issue with NFSv4 and CIFS.

Summary and Conclusion

This document is intended to describe the specification of the Resource Namespace Service, which will be one of the essential services for the realization of a Grid File System. It manages the namespace of federated resources, including virtualized data from file system resources, access control mechanisms, and meta-data management.

This document proposed a set of operations needed to be supported by RNS. Additionally, it proposed two approaches to federation of RNS service instances for scalable, large-scale and distributed namespace management.

Further detailed discussions regarding this specification and the potential evaluation of reference implementations are needed. Additionally, an evaluation should be conducted that examines the aspects of security, performance, consistency, scalability, and reliability. The evaluation needs also to consider functionality of a client library, especially, with and without client attribute cache.

Appendix: WSDL 1.1

The following illustrates the Web Services Description Language (WSDL 1.1) for the Web service methods described in this specification.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="RNS"
    targetNamespace="http://rns.ibm.com"
    xmlns:tns="http://rns.ibm.com"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:gtwsdl="http://www.globus.org/namespaces/2004/01/GTWSDLExtensions"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsrlw=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl"
    xmlns:wsrp=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
    xmlns:wsrpw=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
    xmlns:wsbf=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-01.xsd"
    xmlns:wsntw=
    "http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <!-- RNS Web Service Description File -->
    <wsdl:import
      namespace=
      "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
      location="../wsrf/properties/WS-ResourceProperties.wsdl" />

    <wsdl:import
      namespace=
      "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl"
      location="../wsrf/lifetime/WS-ResourceLifetime.wsdl" />

    <wsdl:import
      namespace=
      "http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
      location="../wsrf/notification/WS-BaseN.wsdl" />

    <types>
      <xsd:schema targetNamespace="http://rns.ibm.com"
           xmlns:tns="http://rns.ibm.com"
           xmlns:xsd="http://www.w3.org/2001/XMLSchema">

           <xsd:import namespace=
             "http://schemas.xmlsoap.org/ws/2004/03/addressing"
             schemaLocation="../ws/addressing/WS-Addressing.xsd" />

           <xsd:import namespace=
             "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-
01.xsd"
             schemaLocation="../wsrf/properties/WS-ResourceProperties.xsd" />

           <xsd:import namespace=
             "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-01.xsd"
             schemaLocation="../wsrf/faults/WS-BaseFaults.xsd" />

           <!-- === RNS Elements Begin === -->

           <xsd:element name="openContext">
```

```
  <xsd:complexType/>
</xsd:element>

<xsd:element name="openContextResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wsa:EndpointReference"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="QueryInput">
  <xsd:sequence>
    <!-- Absolute or relative path, or NULL for current directory -->
    <xsd:element ref="tns:path"          minOccurs="1" maxOccurs="1"/>
    <!-- Starting subentry index for list operations -->
    <xsd:element ref="tns:index"         minOccurs="1" maxOccurs="1"/>
    <!-- Number of subentries to return per list operation message -->
    <xsd:element ref="tns:count"         minOccurs="1" maxOccurs="1"/>
    <!-- If true, resolve all logical and data references per entry -->
    <xsd:element ref="tns:resolve"       minOccurs="1" maxOccurs="1"/>
    <!-- Array of QNames used to indicate what properties to retrieve -->
    <xsd:element ref="tns:propertyTypes" minOccurs="1" maxOccurs="unbound"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ChangeInput">
  <xsd:sequence>
    <!-- Absolute or relative path, or NULL for current directory -->
    <xsd:element ref="tns:path"          minOccurs="1" maxOccurs="1"/>
    <!-- Used only move operation to indicate the new destination -->
    <xsd:element ref="tns:newPath"       minOccurs="1" maxOccurs="1"/>
    <!-- WS-ResourceProperties SetResourceProperties -->
    <xsd:element name="changeProperties" ref="wsrp:SetResourceProperties"
                                         minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="DataReferenceInput">
  <xsd:sequence>
    <!-- Logical or abstract name -->
    <xsd:element ref="tns:logicalName"   minOccurs="1" maxOccurs="1"/>
    <!-- Direct Resource Reference name -->
    <xsd:element ref="tns:physicalName"  minOccurs="1" maxOccurs="1"/>
    <!-- WS-ResourceProperties SetResourceProperties -->
    <xsd:element name="changeProperties" ref="wsrp:SetResourceProperties"
                                         minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<!-- Method Parameters and Returns -->
<xsd:element name="logicalName"   type="xsd:string"/>
<xsd:element name="physicalName"  type="xsd:string"/>
<xsd:element name="path"          type="xsd:string"/>
<xsd:element name="newPath"       type="xsd:string"/>
<xsd:element name="index"         type="xsd:int"/>
<xsd:element name="count"         type="xsd:int"/>
<xsd:element name="resolve"       type="xsd:boolean"/>
<xsd:element name="propertyTypes" type="xsd:QName"/>

<!-- "Context" Object for Maintaining State -->
<xsd:element name="RNSResourceProperty">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:path" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
```

```
        </xsd:element>

    </xsd:schema>
</types>

<!-- Messages -->
<message name="OpenContextRequest">
    <part name="OpenContextRequest" element="tns:openContext"/>
</message>
<message name="OpenContextResponse">
    <part name="OpenContextResponse" element="tns:openContextResponse"/>
</message>

<message name="ListInputMessage">
    <part name="ListInputMessage" type="tns:QueryInput"/>
</message>

<message name="LookupInputMessage">
    <part name="LookupInputMessage" type="tns:QueryInput"/>
</message>

<message name="UpdateInputMessage">
    <part name="UpdateInputMessage" type="tns:ChangeInput"/>
</message>

<message name="CreateInputMessage">
    <part name="CreateInputMessage" type="tns:ChangeInput"/>
</message>

<message name="MoveInputMessage">
    <part name="MoveInputMessage" type="tns:ChangeInput"/>
</message>

<message name="DeleteInputMessage">
    <part name="DeleteInputMessage" element="tns:path"/>
</message>

<message name="ResolveInputMessage">
    <part name="ResolveInputMessage" element="tns:logicalName"/>
</message>

<message name="MapDRInputMessage">
    <part name="MapDRInputMessage" type="tns:DataReferenceInput"/>
</message>

<message name="CreateDRInputMessage">
    <part name="CreateDRInputMessage" type="tns:DataReferenceInput"/>
</message>

<message name="DeleteDRInputMessage">
    <part name="DeleteDRInputMessage" type="tns:DataReferenceInput"/>
</message>

<message name="UpdateDRInputMessage">
    <part name="UpdateDRInputMessage" type="tns:DataReferenceInput"/>
</message>

<!-- === Resource Namespace Service === -->
<portType name="RNSPortType"
    gtwsdl:extends="wsrpw:GetResourceProperty"
    gtwsdl:implements="wsntw:NotificationProducer
                       wsrlw:ImmediateResourceTermination
                       wsrlw:ScheduledResourceTermination"
    wsrp:ResourceProperties="tns:RNSResourceProperty">

    <!-- Operation invoked when creating the web service -->
    <operation name="openContext">
```

```
        <input  message="tns:OpenContextRequest"/>
        <output message="tns:OpenContextResponse"/>
    </operation>

    <!-- WS-ResourceProperties Operations -->
    <operation name="getResourceProperty">
        <input  message="wsrpw:GetResourcePropertyRequest"/>
        <output message="wsrpw:GetResourcePropertyResponse"/>
    </operation>
    <operation name="getMultipleResourceProperties">
        <input  message="wsrpw:GetMultipleResourcePropertiesRequest"/>
        <output message="wsrpw:GetMultipleResourcePropertiesResponse"/>
    </operation>
    <operation name="setResourceProperty">
        <input  message="wsrpw:SetResourcePropertyRequest"/>
        <output message="wsrpw:SetResourcePropertyResponse"/>
    </operation>

    <!-- Lookup Operation -->
    <operation name="lookup">
        <input  message="tns:LookupInputMessage"/>
        <output message="wsrpw:GetMultipleResourcePropertiesResponse"/>
    </operation>

    <!-- List Operation -->
    <operation name="list">
        <input  message="tns:ListInputMessage"/>
        <output message="wsrpw:GetMultipleResourcePropertiesResponse"/>
    </operation>

    <!-- Create Operation -->
    <operation name="create">
        <input  message="tns:CreateInputMessage"/>
        <output message="wsrpw:SetResourcePropertiesResponse"/>
    </operation>

    <!-- Delete Operation -->
    <operation name="delete">
        <input  message="tns:DeleteInputMessage"/>
        <output message="wsrpw:SetResourcePropertiesResponse"/>
    </operation>

    <!-- Update Operation -->
    <operation name="update">
        <input  message="tns:UpdateInputMessage"/>
        <output message="wsrpw:SetResourcePropertiesResponse"/>
    </operation>

    <!-- Move Operation -->
    <operation name="move">
        <input  message="tns:MoveInputMessage"/>
        <output message="wsrpw:SetResourcePropertiesResponse"/>
    </operation>

    <!-- Data Reference Resolve Operation -->
    <operation name="resolve">
        <input  message="tns:ResolveInputMessage"/>
        <output message="wsrpw:GetMultipleResourcePropertiesResponse"/>
    </operation>

    <!-- Data Reference Map Operation -->
    <operation name="mapLogicalDataReference">
        <input  message="tns:MapDRInputMessage"/>
        <output message="wsrpw:SetResourcePropertiesResponse"/>
    </operation>

    <!-- Data Reference Create Operation -->
```

```
        <operation name="createDataReference">
            <input  message="tns:CreateDRInputMessage"/>
            <output message="wsrpw:SetResourcePropertiesResponse"/>
        </operation>

        <!-- Data Reference Delete Operation -->
        <operation name="deleteDataReference">
            <input  message="tns:DeleteDRInputMessage"/>
            <output message="wsrpw:SetResourcePropertiesResponse"/>
        </operation>

        <!-- Data Reference Update Operation -->
        <operation name="updateDataReference">
            <input  message="tns:UpdateDRInputMessage"/>
            <output message="wsrpw:SetResourcePropertiesResponse"/>
        </operation>

    </portType>
</definitions>
```

Author Information

Osamu Tatebe
Grid Technology Research Center, AIST
1-1-1 Umezono, Tsukuba
Ibaraki 3058568 Japan
o.tatebe@aist.go.jp

Manuel Pereira, Leo Luan, Ted Anderson
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120, USA
mpereira@us.ibm.com
leoluan@us.ibm.com
ota@us.ibm.com

Jane Xu
IBM Systems and Technology Group
5600 Cottle Road
San Jose, CA 95193, USA
jxu@us.ibm.com

Intellectual Property Statement

Full Copyright Notice

References

[1] Leo Luan and Ted Anderson, “Grid Namespace for Files”, GGF working draft, GGF8, 2003 https://forge.gridforum.org/projects/gfs-wg/document/Grid_Namespace_for_Files/en/1

[2] S. Shepler, et al., “Network File System (NFS) version 4 Protocol”, RFC3530, 2003

[3] Web Services Addressing (WS-Addressing) http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/

[4] Web Services Resource Properties (WS-ResourceProperties) Version 1.1 03/05/2003 http://www.globus.org/wsrf/specs/ws-resourceproperties.pdf

[SOAP 1.2] http://www.w3.org/TR/soap12-part1/

[State Paper] http://www-106.ibm.com/developerworks/webservices/library/ws-resource/wsmodelingresources.pdf