Resource Namespace Service Specification

Status of This Memo

This memo provides information to the Grid community about resource namespace 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 addressable entity by registering an Endpoint 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 namespace 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 essential resource property definitions that define the Resource Namespace Service.

Contents

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

Introduction

The Resource Namespace Service (RNS) encompasses a multi-faceted approach for addressing the needs of access to resources within a distributed network or grid by way of a universal name that ultimately resolves to a meaningful address, with a particular emphasis on hierarchically managed names that may be used in human interface applications.

RNS is intended to facilitate namespace services for a wide variety of Grid applications and 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.

The practical necessity of conveniently accessing the growing number of Web services, corresponding applications, service artifacts and other service resources, has manifest an escalating need for a generalized resource namespace service. Additionally, the ever-increasing appreciation for resource virtualization has amplified the benefits of this service, which is capable of maintaining a name to multi-address mapping, since the namespace thereby virtualizes all endpoint references or resource addresses.

The Resource Namespace Service utilizes document style messaging that takes advantage of XML, avoids unnecessary constraints (such as inflexible operation parameters and rigid return types), is fully WSRF-compliant, and allows for extensibility via resource property profiling. This document proposes a set of document style operations exploiting well-defined resource properties that define the RNS service.

The RNS specification document has emerged from the Grid File System Working Group (GFS-WG); principally based on the Virtual Filesystem Directory Service (VFDS) specification from that group. Two major deliverables of the WG are (1) architecture of Grid File System Services and (2) specification of namespace services. The VFDS specification was intended to address (2) by proposing a namespace service that would easily satisfy the rudimentary need of managing a namespace of federated and virtualized data, access control mechanisms, and a minimal set of associated meta-data [1]. As the specification matured, it became more and more obvious that a generalized namespace service would have substantial application in a wide variety of Grid services. Consequently, the filesystem and data specific features of VFDS have been factored out of this specification, yielding a generic resource namespace service that is no longer tailored to data related applications. However, RNS features an extensible design allowing normative profile specifications, such as OGSA Basic Profiles [5], to define a standard set of resource properties for specific instantiations of the namespace service. For this reason, this document will not address any data related namespace requirements but will initially include, in the appendix (see Appendix: Grid File System Profile), a proposed profile for Grid File System instantiations of RNS.

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.

Notational Conventions
The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119.

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 *human interface names*, *virtualized reference names*, and *endpoint references*.

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 interface names* directly to *endpoint references* or resource reference addresses. Since the properties of the *endpoint 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 or resource virtualization. A second level of indirection may be appreciated when mapping *human interface names* to *virtualized references* (identified by *logical* or *abstract* names), which in turn map *logical names* to *endpoint references* and hence the second level of indirection. The advantage of using a *logical name* to represent a *virtualized reference* is that *logical names* may be referenced and resolved 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. Although the RNS specification includes an optional port type that services virtualized resource to endpoint resolution, as an independent service, it is not required that clients use this RNS resolution service, since the logical name can potentially be resolved by a separate logical to endpoint resolution service. In contrast, note that mapping information and associated pointer handles for directly mapped *human interface name* to *endpoint references* are not exposed by RNS and are therefore only used internally by RNS.

Following is a diagram that illustrates the three-tier naming architecture; please note that this diagram is strictly intended to illustrate the levels of the naming architecture and is not limited to the types of references shown:

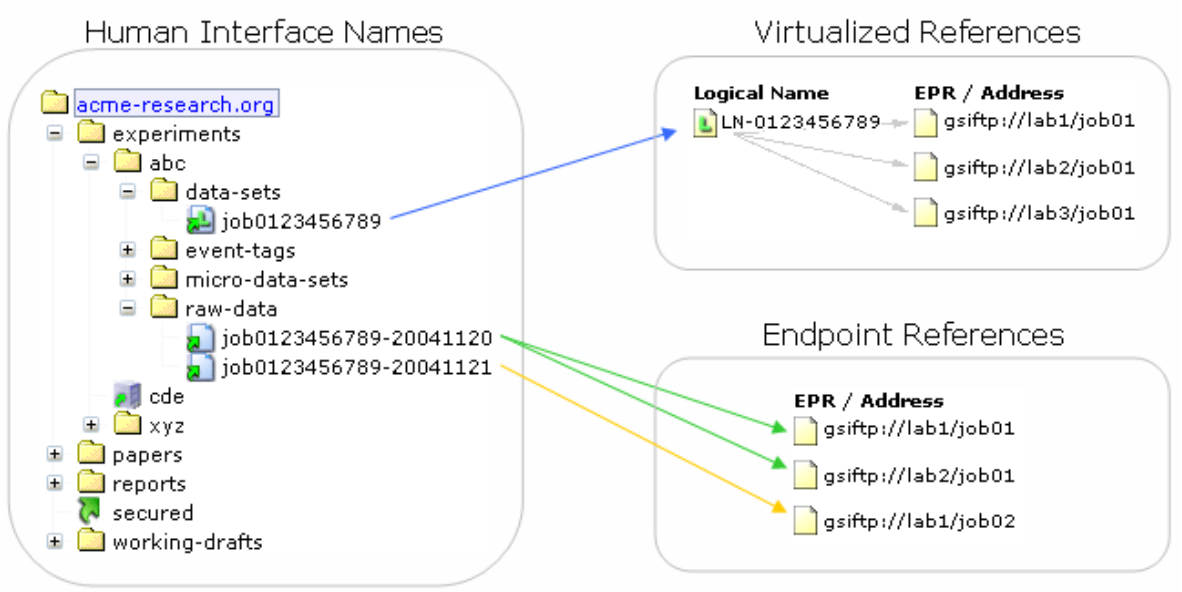

Figure 1 - Three-Tier Naming Architecture

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.

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, copies of the same resource, or operationally identical services. A description of each follows:

1.1.1 Virtual Directories

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 virtual directory may be considered analogous to a *collection*, *category*, or *context*—to the extent that these terms are used in most directory or catalogue contexts. Virtual directories do not have any time or space existence outside of the namespace and strictly serve to facilitate hierarchy, and thus categorization, by presenting the illusion of compartments, which may contain sub-compartments as well as junctions.

Corresponding resource property QName = *VirtualDirectory*

1.1.2 Junctions

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 four basic types: *virtualized references*, *endpoint references*, *referrals*, and *aliases*.

1.1.2.1 Virtualized Reference Junction

A *virtualized reference junction* is a junction that either contains an endpoint reference (EPR) or universal resource locator (URL) that points to a secondary service, like a Replica Location Service (RLS), for name-to-address resolution given a context unique (potentially global) *logical name*. This specification does not mandate a required format for the target property value of a virtualized reference. In other words, the format of the EPR is not mandated.

This RNS specification includes the description of a non-hierarchical name-to-address resolution service, defined in an independent port type that facilitates simple *logical name* resolution as an optional adjunct service. (see *RNS Resolver Service*)

Corresponding resource property QName = *LogicalReference*

1.1.2.2 Endpoint Reference Junction

An *endpoint reference junction* is an entry that maps to at least one Web or Grid resource by way of a WS-Addressing[3] Endpoint 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.

Corresponding resource property QName = *EPR*

1.1.2.3 Referral Junction

Referral junctions are junctions that link to other RNS instances, thereby facilitating such features as *symbolic links* (or *soft links*), 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.acme-research.org/.

Corresponding resource property QName = *Referral*

1.1.2.4 Alias Junction

An *alias junction* is a junction that references another entry within the same service instance to provide the feature of representing a single entry in multiple locations in the namespace hierarchy or simply by multiple names; this effect is comparable to the conventional Unix filesystem hard links.

Since an *alias junction* is intended to represent the *entry* it points to, the service implementation MUST NOT allow an *entry* to be deleted if one or more *alias junctions* point to it. Therefore, if an *entry's AliasCount* property is greater than one, it may not be deleted. Optionally, the service MAY allow an *entry* to be deleted if it has one or more *alias junctions* pointing to it, if and only if it dynamically reassigns all of the *entry's* properties to one of the *alias junctions*, thereby transforming the elected *alias junction* into a basic *entry*. *Alias junction* election in this context is not mandated by this specification.

Corresponding resource property QName = *Alias*

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 service.

1.2 Document Style Messaging

RNS exploits a document style message exchange approach to services. In so doing, it offers useful features whose benefits are beyond the flexibility of traditional remote procedure call (RPC) style services. In this approach RNS leverages the capabilities of XML to communicate messages that may be tailored according to the request. Additionally, greater flexibility is realized in the exchange of parameters and complex types or objects. A document style interface facilitates a greater extensibility of the service without breaking calling applications.

Access to RNS entry metadata is 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.

1.2.1 WSRF Compliant Service

In addition to a document style interface, RNS provides standard access and manipulation of stateful resource properties via Web Service Resource Framework (WSRF). The RNS interface 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 an *IteratorContext*.

The RNS *IteratorContext* resource is designed specifically for the purpose of maintaining stateful properties related to iterative operations. This is particularly necessary when listing a potentially large directory, since the application may not want to have all of the subentries returned in a single message and therefore may request to receive the list in segments. To ensure each segment is internally consistent within a projected list, the RNS service MUST support a point-in-time result-set reflecting the entire list at the time the initial *list* request was processed. The *IteratorContext* then enables subsequent *list* requests to be made that retrieve segment by segment from the point-in-time result-set maintained on the service end. Consequently, *IteratorContexts* are automatically constructed for every list request and SHOULD be destroyed after the iterator has been exhausted. The resource properties document associated with the *IteratorContext* resource is described in further detail in the next section.

RNS implements the following WSRF standard operations GetMultipleResourceProperties message exchange for all query oriented operations and the SetMultipleResourceProperties message exchange for all change oriented operations.

The RNS port type (RNSPortType) extends the GetResourceProperty port type defined by WS-ResourceProperties[3], implementing the GetResourceProperty, GetMultipleResourceProperties, and SetResourceProperty operations. Additionally, for lifetime management, the RNS port type also implements the Destry, CurrentTime, TerminationTime, and SetTerminationTime.

1.2.2 Resource Properties Documents

A resource properties document is the XML document representing a logical composition of resource property elements for a given resource.[4]

1.2.2.1 RNS IteratorContext – The WS-Resource

As described in section 1.2.1, RNS defines a stateful resource referred to as an *IteratorContext*. The following resource properties MUST be supported and available in the WS-Resource message exchange:

QName	Description
ChildCount	Integer value that denotes the number of subentries found in the current directory being listed.
DirectoryPath	String representing the full path of the current directory being listed.
IteratorContextID	String value that denotes the resource identifier of the *IteratorContext* WS-Resource. The value SHOULD be considered transient and only unique in its corresponding service instance for the lifetime of the resource.
IteratorIndex	Integer representing the current index or marker corresponding to a current iterator operation; can be queried between iterator messages. [default value is "**0**"]

Following is the resource properties document associated with the RNS *IteratorContext* WS-Resource.

```
<!-- "Context" Resource for Maintaining State -->
<xsd:element name="IteratorContext">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:childCount"        minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="tns:directoryPath"     minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="tns:iteratorIndex"     minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

The following simply lists the resource property element declarations referred to by the resource properties document above.

```
<!-- Resource property element declarations -->
<xsd:element name="childCount"        type="xsd:int"/>
<xsd:element name="directoryPath"     type="xsd:string"/>
<xsd:element name="iteratorIndex"     type="xsd:int"/>
```

1.2.2.2 Resource Properties for Namespace Entries

The previous section describes the resource properties document associated with the WS-Resource of the RNSService port type, used for stateful communication. RNS facilitates access and manipulation of namespace *entries* by way of document style messaging. As indicated in the description of the RNS WS-Resource, the standard WSRF operations do not involve directly accessing or modifying namespace *entries* but rather an RNS *IteratorContext*.

RNS specifies two fundamental service objects: (1) the first is the RNS *IteratorContext* resource, which was described in section 1.2.2.1, and (2) the second is a namespace component referred to as an RNS or namespace *entry*. Each *entry* represents a namespace node that symbolizes either a *virtual directory* or a *junction* (see *Basic Namespace Components 1.1*).

Information about namespace *entries* is exchanged using document style messaging rather than RPC style object serialization. We only refer to *entries* as "objects" in a conceptual manner, understanding that they are not classes that will be instantiated in the client runtime environment. For this reason this specification does not define an object or complex type that can be acted on directly by any application. Instead, the specification will exhibit a profile approach by defining the static list of resource properties corresponding to the namespace *entry* object or resource.

1.2.2.2.1 Required Entry Properties

All of the following properties MUST be implemented to represent properties of a namespace *entry* by an RNS service implementation. Please notice that each of the following namespace *entry* properties SHOULD be considered to represent transient values.
(*Entry* signifies a runtime instance of a valid namespace entry object.)

QName	Description
Basic Properties	
AliasCount	Integer: Number of known aliases of *Entry*
ChildCount	Integer: Number of subentries corresponding to *Entry*, if and only if *Entry* is a *Virtual Directory*; zero or NULL otherwise.
Description	String: Description of *Entry*
ModificationTime	DateTime (`xsd:dateTime`) representation of the last modified timestamp of *Entry*
Name	String: Representation of the human interface name of *Entry*
Type	String: Value denoting a type of *entry*; valid values are: *LogicalReference*, *EPR*, *Alias*, *Referral*, and *VirtualDirectory.* (which are also the "local part" values of the respective QNames)
Reference Properties – Properties that host target information	
EPR	String: Used to set or add a single Endpoint Reference
EPRs	String: Used to retrieve all Endpoint References associated with *Entry*
LogicalReference	String: Used to set or add a single Logical Reference
LogicalReferences	String: Used to retrieve all Logical References associated with *Entry*
TargetPath	String: Absolute path that identifies the target *entry* of an *Alias* junction.

1.2.2.2.2 Extensible Entry Properties

In addition to the well-defined properties for namespace *entries*, an RNS service MUST implement operations that enable administrative applications to add and remove user-defined properties that may correspond to a profile definition. Thus the resource properties document design is extensible in that user-defined properties can be added and removed without requiring modification of the core service. (See section 1.3.4)

1.2.2.2.3 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 MUST however enforce such relationship requirements on the service side, but a good understanding of what correct property relationships are is helpful.

1.2.2.3 Properties for Operation Parameters

In addition to *IteratorContext* and *Entry* resource properties, an RNS service MUST implement the following properties and accommodate their use in the designated service operations listed in section 1.3.

QName	Description
All	Boolean: Used in place of enumerating all of the available properties (signified by QNames) of a given resource
AutoChangeDir	Boolean value that if “true” will cause the current working directory or *BaseDirectory* to change to the directory being listed. [default value is “**true**”]
AutoResolve	Boolean value that if “true” will cause this operation to attempt to resolve any virtualized resources by their logical name using the companion RNS Resolver Service. Only one level of resolution is required, so if a logical reference resolves to a virtualized address/reference only the first level of abstraction is resolved. [default value is “**false**”]
Name	String: Simple character string representation of a context dependant name property.
Path	String: Value representing a path or sequence of hierarchical tree levels in the namespace tree; used as a generic parameter property for most operations. Generally represents the only globally unique persistent namespace entry identifier.
BaseDirectory	String: Fully qualified path of the current working directory or *BaseDirectory* with which construction of an absolute path may be realized by concatenating the *BaseDirectory* with a relative *entry* name. This property MUST be exchanged in all query operations (see 1.3.1.3) and can therefore be leveraged for use in input parameters.
EndOfList	Boolean: Value that if “true” indicates an iterative list operation has reached the end of the list. This property MUST be exchanged in all query operations (see 1.3.1.3).

1.3 Operations of the Resource Namespace Service

RNS is composed of the following types of operations:

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

To retrieve information about a particular namespace *entry*, a standard message exchange (operation) is initiated by a message request containing a list of all of the property names (QNames) whose values are to be retrieved. The operation 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 comprised of nest-able message elements.

1.3.1 Operation Parameters

Please note that in the current WSRF implementation by Globus 3.9.4, only one parameter is permitted per operation. Before examining the purposed operations, it is necessary to review the associated operation parameters. All RNSService port type operations take one of the following input parameters.

1.3.1.1 QueryInput

This is a document literal service compliant message (complexType) that contains two elements:

Parameter Name	*Description*
parameterList	A complexType that encapsulates an unbound array of name-value pairs
propertyTypes	An unbound array of (xsd:QName) strings

```
<xsd:complexType name="QueryInput">
  <xsd:sequence>
    <!-- Dynamic list of parameters -->
    <xsd:element ref="tns:parameterList" 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 message (complexType) that contains two elements:

Parameter Name	*Description*
parameterList	A complexType that encapsulates an unbound array of name-value pairs
changeProperties	A `wsrp:SetResourceProperties` message [4]

```
<xsd:complexType name="ChangeInput">
  <xsd:sequence>
    <!-- Dynamic list of parameters -->
    <xsd:element ref="tns:parameterList" minOccurs="1" maxOccurs="1"/>
    <!-- WS-ResourceProperties SetResourceProperties -->
    <xsd:element name="changeProperties" ref="wsrp:SetResourceProperties"
                                         minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

The *changeProperties* message allows the processing of a single request message to make multiple changes to the target resource properties document. There are three types of changes:

- *Insert*: wherein a new property element is inserted into the resource properties document
- *Update*: wherein existing property element(s) are modified
- *Delete*: wherein an existing property element(s) are removed

Therefore, property values MUST be sent using the appropriate change type for the request. In other words, if the caller desires to add a new property value to a given resource they must set the value in the *Insert* element.

The format of this request message MUST be:

```
<wsrp:SetResourceProperties>
 {
  <wsrp:Insert >
    {any}*
  </wsrp:Insert> |
  <wsrp:Update >
    {any}*
  </wsrp:Update> |
  <wsrp:Delete ResourceProperty=”QName” />
 }+
</wsrp:SetResourceProperties>
```

1.3.1.3 QueryResponse

This is a document literal service compliant message (complexType) that contains three components: *BaseDirectory*, *EndOfList*, and an array of unrestrained message elements. The following is the WSDL representation of the *QueryResponse*:

```
<xsd:complexType name="QueryResponse">
  <xsd:sequence>
    <xsd:element ref="tns:baseDirectory" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="tns:endOfList" minOccurs="1" maxOccurs="1"/>
    <xsd:any minOccurs="0" maxOccurs="unbound"/>
  </xsd:sequence>
</xsd:complexType>
```

1.3.1.4 ChangeResponse

This is a document literal service compliant message (complexType) that is a void message:

```
<xsd:complexType name="ChangeResponse" />
```

1.3.2 Namespace Operations

The following is a comprehensive list of operations defined in the RNS namespace port type (RNSPortType) specification.

1.3.2.1 create

Enables an application 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 by the service host. This operation is primarily used for the creation of namespace entries, but may also effect the creation of other datastore objects (like Endpoint Reference entries if the service implementation utilizes a separate entry for storing EPR information).

Parameter: *ChangeInput* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)

There are no changeProperties used in this operation.

Path MUST be specified in the parameterList of *ChangeInput*.
(for values see 1.2.2.3):

QName	Description
Path	The absolute path of the *entry* to be created.
Name	String representation of the human interface name of *Entry.* **Optional** MAY be used as an appendix to the value of *Path*.

Exactly one type (*LogicalReference, EPR, Alias, Referral*, or *VirtualDirectory*) MUST be specified.

The following entry properties MAY be specified in the parameterList of *ChangeInput*
(for values see 1.2.2.2.1):

QName	Description
Description	String: Optional description
ModificationTime	DateTime (xsd:dateTime) representation of the last modified timestamp
Type	String: Value denoting a type of *entry*; valid values are: *LogicalReference*, *EPR*, *Alias*, *Referral*, and *VirtualDirectory*
EPR	Value of a single Endpoint Reference to be associated with *Entry*
LogicalReference	Value of a single Logical Reference to be associated with *Entry*
TargetPath	The absolute path of the target *entry*. Set only if *Entry* is an *Alias*.
Any adjunct resource property QNames and respective values set at runtime. See 1.3.4	

Note that more than one *EPR* and *LogicalReference* elements MAY be included in a single message exchange, effectively representing a list of values.

1.3.2.2 delete

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

Parameter: *ChangeInput* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)

The following parameter MUST be specified in the parameterList of *ChangeInput*.
(for values see 1.2.2.3):

QName	Description
Path	The absolute path of *Entry* to be deleted.

There are no changeProperties used in this operation.

1.3.2.3 list

Enables an application to submit a request message that contains an array of property names whose values are 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. Since directories may contain a very large number of subentries, this operation enables the caller to specify the maximum number of subentries allowable per message exchange (*IteratorMaxAtOnce*). If this parameter property is specified as a non-zero value, then an *IteratorContext* will automatically be constructed and returned to the caller using standard WSRF mechanisms for stateful resource interaction.

Parameter: *QueryInput* (see 1.3.1.1)
Returns: *QueryResponse* (see 1.3.1.3)

The following parameter(s) MAY be specified in the parameterList of *QueryInput*. *Path* value MUST be specified.
(for values see 1.2.2.1 & 1.2.2.3):

QName	Description
Path	The absolute path of the *virtual directory* to list. **Required**.
AutoChangeDir	Boolean value that if "true" will cause the current working directory or *BaseDirectory* to change to the directory being listed. [default value is "**true**"]
AutoResolve	Boolean value that if "true" will cause this operation to attempt to resolve any virtualized resources (one level) by their logical name using the companion RNS Resolver Service. [default value is "**false**"]
IteratorMaxAtOnce	Integer indicating the maximum number of entries allowed in a single message; used in iterative list operations. A value of zero "0" indicates no maximum limit. [default value is "**0**"]

(continued on next page)

*(continued from previous page – **list** operation)*

At least one entry property type MUST be specified for this operation (see below).

The following entry properties MAY be specified in the `propertyTypes` of *QueryInput* (for values see 1.2.2.2.1):

QName	Description
All	Used in place of enumerating all of the available properties (signified by QNames); indicates ALL properties should be returned.
AliasCount	Number of known aliases of *Entry*
ChildCount	Number of subentries corresponding to *Entry*, if and only if *Entry* is a *Virtual Directory*; zero or NULL otherwise.
Description	Optional description of *Entry*
ModificationTime	DateTime (`xsd:dateTime`) representation of the last modified timestamp of *Entry*
Name	String representation of the human interface name of *Entry*
Type	String value denoting the type of *entry*; valid values are: *LogicalReference*, *EPR*, *Alias*, *Referral*, and *VirtualDirectory*
EPR	Used to set or add a single Endpoint Reference
EPRs	Used to retrieve all Endpoint References associated with *Entry*
LogicalReference	Used to set or add a single Logical Reference
LogicalReferences	Used to retrieve all Logical References associated with *Entry*
TargetPath	The absolute path of the target entry; if and only if *Entry* is an *Alias*; empty or NULL otherwise.
Any adjunct resource property QNames and respective values set at runtime. See 1.3.4	

1.3.2.4 lookup

Enables an application 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* (see 1.3.1.1)
Returns: *QueryResponse* (see 1.3.1.3)

The following parameter(s) MAY be specified in the parameterList of *QueryInput*. *Path* value MUST be specified.
(for values see 1.2.2.1 & 1.2.2.3):

QName	Description
Path	The absolute path of the *entry* to lookup. **Required**.
AutoResolve	Boolean value that if "true" will cause this operation to attempt to resolve any virtualized resources by their logical name using the companion RNS Resolver Service.

At least one entry property type MUST be specified for this operation (see below).
The following entry properties MAY be specified in the propertyTypes of *QueryInput*
(for values see 1.2.2.2.1):

QName	Description
All	Used in place of enumerating all of the available properties (signified by QNames); indicates ALL properties should be returned.
AliasCount	Number of known aliases of *Entry*
ChildCount	Number of subentries corresponding to *Entry*, if and only if *Entry* is a *Virtual Directory*; zero or NULL otherwise.
Description	Optional description of *Entry*
ModificationTime	DateTime (`xsd:dateTime`) representation of the last modified timestamp of *Entry*
Name	String representation of the human interface name of *Entry*
Type	String value denoting the type of *entry*; valid values are: *LogicalReference*, *EPR*, *Alias*, *Referral*, and *VirtualDirectory*
EPR	Used to set or add a single Endpoint Reference
EPRs	Used to retrieve all Endpoint References associated with *Entry*
LogicalReference	Used to set or add a single Logical Reference
LogicalReferences	Used to retrieve all Logical References associated with *Entry*
TargetPath	The absolute path of the target entry; if and only if *Entry* is an *Alias*; empty or NULL otherwise.
Any adjunct resource property QNames and respective values set at runtime. See 1.3.4	

1.3.2.5 update

Enables an application 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* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)

The following parameter MUST be specified in the parameterList of *ChangeInput*.
(for values see 1.2.2.3):

QName	Description
Path	The absolute path of the *entry* to update. **Required**.

The following entry properties MAY be specified in the changeProperties of *ChangeInput*
(for values see 1.2.2.2.1):

QName	Description
Description	Optional description
ModificationTime	DateTime (`xsd:dateTime`) representation of the last modified timestamp
Name	String representation of the human interface name of *Entry*
Path	The absolute path the *entry* should be changed to. Used in "move" operations
EPR	Add a single Endpoint Reference to be associated with *Entry*
LogicalReference	Add a single Logical Reference to be associated with *Entry*
TargetPath	Set the absolute path of the target entry. Set only if *Entry* is an *Alias*.
Type	String: Value denoting a type of *entry*; valid values are: *LogicalReference*, *EPR*, *Alias*, *Referral*, and *VirtualDirectory*
Any adjunct resource property QNames and respective values set at runtime. See 1.3.4	

Note that more than one *EPR* and *LogicalReference* elements MAY be included in a single message exchange, effectively representing a list of values.

The *ChangeInput* parameter is fully capable of inserting, updating, and deleting properties in a single message exchange via the *changeProperties* component. Values MUST be represented by the appropriate change type: Insert, Update, or Delete. (see section 1.3.1.2)

1.3.3 Implicit Operations

This specification attempts to maximize the flexible capabilities of document style messaging while maintaining a simple, clearly defined API. Unlike traditional RPC based approaches, RNS utilizes a minimal set of operations used for exchanging messages that are potentially capable of performing multiple tasks in a single exchange. Rather than defining a separate operation for each task, this specification describes a number of implicit operations, which are essentially descriptions of how to perform conventional directory service tasks using the well defined service operations.

1.3.3.1 move

Move a namespace *entry* from one location in the hierarchical namespace tree to another.

Operation: *update* (see 1.3.2.5)

The following parameter MUST be specified in the parameterList of *ChangeInput*.
(for values see 1.2.2.3):

QName	Description
Path	The absolute path of the *entry* to update. **Required**.

The following properties MUST be specified in the changeProperties of *ChangeInput*
(for values see 1.2.2.2.1):

QName	Description
Path	The absolute path the *entry* should be changed to. Used in "move" operations. Value MUST be expressed in the *Update* element of changeProperties.

1.3.3.2 rename

Rename a namespace *entry*.

Operation: *update* (see 1.3.2.5)

The following parameter(s) MAY be specified in the parameterList of *ChangeInput*. *Path* MUST be specified.
(for values see 1.2.2.3):

QName	Description
Path	The absolute path of the *entry* to update OR the BaseDirectory if *Name* is specified. **Required**.
Name	String representation of the human interface name of *Entry.* **Optional:** MAY be used to denote the subentry relative to the value of *Path*.

At least one of the following properties MUST be specified in the changeProperties of *ChangeInput*
(for values see 1.2.2.2.1):

QName	Description
Name	String representation of the human interface name of *Entry*. Used only if a *BaseDirectory* is specified and the value of the *Name* input parameter is non-NULL.
Path	The absolute path denoting the new path/name of the *entry*.

1.3.3.3 mkdir

Make a directory *entry* in the namespace; a *virtual directory*.

Operation: *create* (see 1.3.2.1)

The following parameters MUST be specified in the parameterList of *ChangeInput*.
(for values see 1.2.2.3 and 1.2.2.2.1):

QName	Description
Path	The absolute or relative path of the *virtual directory* to create.
Type	Set with a value of *VirtualDirectory*.

1.3.4 Profile Extension Operations

RNS features an extensible design allowing normative profile specifications, such as OGSA Basic Profiles [5], to define a standard set of resource properties for specific instantiations of the namespace service. This feature facilitates extensibility without requiring modification to the RNS specification or implementation, eliminates the necessity to draft a design specification, and eliminates the necessity to develop any implementation code that "extends" or "subclasses" any RNS component. Traditional software engineering practices generally extend a service class or component by subclassing it and adding specific functionality tailored for a particular purpose. This approach usually requires that each time a new function is added, software development and deployment is necessary.

In an effort to leverage the flexibility and abstractness of document style Web services, RNS proposes a mechanism that facilitates dynamic runtime extensibility with the use of adjunct resource properties. These adjunct resource properties may be defined by a Basic Profile [5]. An adjunct resource property may be added to the effectual resource properties document of the RNS entry properties document. This means that an administrator of the RNS service may define resource properties that will be used in addition to the *required entry properties* [Required Entry Properties 1.2.2.2.1], thereby effectively augmenting the representation of the RNS Entry resource and extending the resource properties associated with it to include the newly added adjunct resource properties.

The RNS service MUST be able to support use of any dynamically added adjunct resource property, by properly allowing the use of message elements identified by QNames that represent the adjunct resource property. Values MUST be expressible in XML compatible data types [6].

1.3.4.1 Profile Extension Operation Parameters

The following table defines the properties used as parameters in the RNS profile extension operations.

QName	Description
DataType	WSDL compatible representation of the XML data type. Possible values are: *string, boolean, base64Binary, hexBinary, float, decimal, double, anyURI, QName, duration, dateTime, time,* and *date*. Example: "string" See [6].
Description	String description of the adjunct resource property
Name	Name of the adjunct resource property, serving as the QName (local part) used to represent the property in general operations.
Profile	Optional string value denoting the Profile this adjunct resource property is associated with.

Following are the operations that enable management of adjunct resource properties defined in the RNS port type (RNSPortType) specification:

1.3.4.2 deleteProperty

Delete an existing *adjunct resource property* from the registry. This operation will delete ALL instances of the property even if more than one entry has stored values corresponding to the property.

Parameter: *ChangeInput* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)

The changeProperties of *ChangeInput* is not used in this operation.

The following properties MUST be specified in the parameterList of *ChangeInput* (for values see 1.3.4.1)::

QName	Description
Name	Name of the *adjunct resource property* to be deleted. (QName)

1.3.4.3 insertProperty

Store a new *adjunct resource property* to the registry. An exception is thrown if the *adjunct resource property* specified already exists in the service's persistent database.

Parameter: *ChangeInput* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)

The changeProperties of *ChangeInput* is not used in this operation.

The following properties MUST be specified in the parameterList of *ChangeInput* (for values see 1.3.4.1):

QName	Description
Name	Name of the *adjunct resource property* to be inserted. (QName)

The following properties MAY be specified in the parameterList of *ChangeInput* (for values see 1.3.4.1):

QName	Description
DataType	WSDL compatible representation of the XML data type.
Description	Description of the *adjunct resource property*
Profile	Optional string value denoting the Profile this *adjunct resource property* is associated with.

1.3.4.4 listProperties

Lists all currently registered *adjunct resource properties*.

Parameter: *QueryInput* (see 1.3.1.1)
Returns: *QueryResponse* (see 1.3.1.3)

The following parameter(s) MAY be specified in the parameterList of *QueryInput* serving as query filters.
(for values see 1.3.4.1):

QName	Description
DataType	Use as a list filter. Only *adjunct resource properties* that match the value of this parameter will be returned.
Name	Use to identify a specific *adjunct resource property* to list. Only the *property* that matches the value of this parameter will be returned.
Profile	Use as a list filter. Only *adjunct resource properties* that match the value of this parameter will be returned.

At least one property type MUST be specified for this operation (see below).
The following properties MAY be specified in the propertyTypes of *QueryInput* to specify what properties of the returning *adjunct resource properties* should be listed.
(for values see 1.3.4.1):

QName	Description
DataType	WSDL compatible representation of the XML data type.
Description	Description of the *adjunct resource property*
Name	Name of the *adjunct resource property*. (QName)
Profile	String value denoting the Profile this *adjunct resource property* is associated with.

1.3.4.5 updateProperty

Updates an existing *adjunct resource property*.

Parameter: *ChangeInput* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)

The following properties MUST be specified in the parameterList of *ChangeInput*.
(for values see 1.3.4.1):

QName	Description
Name	Name of the *adjunct resource property* to be updated. (QName)

The following properties MUST be specified in the changeProperties of *ChangeInput*
(for values see 1.3.4.1):

QName	Description
DataType	WSDL compatible representation of the XML data type.
Description	Description of the *adjunct resource property*
Name	Name of the *adjunct resource property*. (QName)
Profile	String value denoting the Profile this *adjunct resource property* is associated with.

The *ChangeInput* parameter is fully capable of inserting, updating, and deleting properties in a single message exchange via the *changeProperties* component. Values MUST be represented by the appropriate change type: Insert, Update, or Delete. (see section 1.3.1.2)

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 consider an architecture that federates multiple namespace servers in a hierarchical fashion. Similar to the well established DNS model, RNS service providers 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, a fundamental namespace service that is capable of addressing a wide variety of namespace related needs from virtualized services and artifacts to federated global data.

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: Grid File System Profile

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 a file system namespace service. File system directory services will manage the namespace of federated and virtualized data from file system resources [1]. It will provide features such as (a) virtualized hierarchical namespaces for files or potentially other types of data (such as live data feeds), (b) efficient and transparent file sharing, and (c) ability to describe and manage file-system and application-specific metadata.

This document appendix intends to present a standard profile, for use with RNS, that describes a Virtual Filesystem Directory Service (VFDS) specification. It proposes a list of resource properties needed to be supported by file system directory services.

The following table presents a set of resource properties that MUST be supported for file system directory service applications.

QName	Description
Checksum	String representation of the actual checksum corresponding to the physical file or fileset symbolized by this data resource junction.
ChecksumType	String representation of the checksum type or algorithm used to produce the checksum.
Complete	Identifies whether or not the file or filesystem source targeted by this VFDS entry is complete. In the case of files, a value of true connotes all of the file content is embodied in the file; for filesets (filesystem subtrees) this identifies whether or not the fileset is complete in terms of number of files participating and the coherency of these files.
MutableSource	Identifies whether or not the file or filesystem source targeted by this VFDS entry can change.
ReadOnly	Identifies whether or not a local copy of the data should be locally read-only.
ReplicaCopy	Identifies whether or not the file or filesystem source targeted by this VFDS entry is a replica copy.
Size	The physical size of the targeted data source. If the target data is in the form of a file (implying a PFN) then this value discloses the size of the file in bytes. If the target data is in the form of a fileset (implying a PFSN) then this value discloses the summation size of all the contained files.
Timestamp	The replica or fileset's point-in-time timestamp corresponding to the time at which the source snapshot was made.
Version	The version number of the targeted data if available.

2 Appendix: Resource Resolution Service

The Resource Resolution Service, which will henceforth be referred to as RNS Resolver, is a companion service to RNS providing operations that enable management and resolution of *virtualized references*. The RNS Resolver service is independent of RNS, and RNS is independent of it. RNS Resolver MAY be used by RNS and other services and applications, at the same service URL as the RNS namespace service, using a different port type (RNSResolverPortType).

As described in the RNS specification for namespace services, RNS Resolver only addresses the second and third tiers of the overall naming scheme—that is the level of strictly mapping *logical names* to *endpoint references*.

2.1 RNS Resolver Basic Components

RNS Resolver is comprised of two fundamental service components: *logical names* and *endpoint references* or *addresses*. These two basic components, also referred to as *virtualized reference*s, are used to serve a name-to-address resolution service, capable of a many-to-many mapping between names and addresses. This service does not maintain any complex relationships between components, but rather an intuitive mapping of *logical names* to *endpoint references.* One *logical name* maps to at least one *endpoint reference*, but is unbound regarding the number of targets allowable. It is also possible that a given *endpoint reference* is referenced by more than one *logical name*. A description of each follows:

2.1.1 Logical Reference

A Logical Reference is characterized by its *logical name*, which is a logically unique—potentially globally unique—identifier of some resource. A *logical name* does not have any intrinsic value nor is it meaningful outside of the context for which it is intended—it is simply a unique name that is used to identify a resource or set of resources that have been logically virtualized. *Logical names* may be used in registries other than RNS Resolver and can potentially be interoperable amongst different resolution services.

2.1.2 Endpoint Reference

An Endpoint Reference in the context of Web services is fundamentally a formatted reference string, usually represented in XML, that targets a referenceable entity, processor, or resource where Web service messages can be exchanged. Endpoint References convey the information needed to identify/reference a Web service endpoint.[3]

2.2 Document Style Messaging

RNS Resolver exploits a document style message exchange approach to services.
(Please refer to section 1.2)

2.3 Operations of RNS Resolver

RNS Resolver is composed of the following operations:

1) An operation for resolving *logical names* to *endpoint references*.
2) Operations for creating, removing, and updating *virtualized references*.

To retrieve information about a particular *virtualized reference*, a standard message exchange (operation) is initiated by a message request containing a list of all of the property names (QNames) whose values are to be retrieved. The operation 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 comprised of nest-able message elements.

2.3.1 Operation Parameters

Please refer to section 1.3.1 for additional property definitions.

QName	Description
Description	String description of either a *logical name* or *endpoint reference*
EPR	Used to set or add a single Endpoint Reference value
EPRs	Used to retrieve an inclusive list of Endpoint References mapped by a given *logical name*
LogicalReference	Used to set or add a single Logical Reference value according to its *logical name*
LogicalReferences	Used to retrieve an inclusive list of Logical References mapped by a given *logical name*

2.3.2 RNS Resolver Operations

The following is a comprehensive list of operations defined in the RNS Resolver port type (RNSResolverPortType) specification.

2.3.2.1 deleteEndpointReference

Delete an existing *endpoint reference* from all mappings, unless it represents the only *endpoint reference* mapped by a given *logical name* in which case an exception is thrown.

Parameter: *ChangeInput* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)

The changeProperties of *ChangeInput* is not used in this operation.

The following properties MUST be specified in the parameterList of *ChangeInput* (for values see 2.3.1):

QName	Description
EPR	The Endpoint Reference to be deleted

2.3.2.2 deleteLogicalReference

Delete an existing *logical name* to *endpoint reference* mapping.

Parameter: *ChangeInput* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)

The changeProperties of *ChangeInput* is not used in this operation.

The following properties MUST be specified in the parameterList of *ChangeInput* (for values see 2.3.1):

QName	Description
LogicalReference	The *logical name* of the Logical Reference to delete

2.3.2.3 insertLogicalReference

Store a new *logical name* to *endpoint reference* mapping. An exception is thrown if the *logical name* used already exists in the service's persistent database.

Parameter: *ChangeInput* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)

The changeProperties of *ChangeInput* is not used in this operation.

The following properties MUST be specified in the parameterList of *ChangeInput* (for values see 2.3.1):

QName	Description
LogicalReference	The *logical name* of this Logical Reference
EPR	The single Endpoint Reference to be mapped

The following properties MAY be specified in the parameterList of *ChangeInput* (for values see 2.3.1):

QName	Description
Description	Description of the Logical Reference

Note that a message MAY contain multiple *EPR* elements, which effectively represents a list.

2.3.2.4 resolve

Takes a *logical name* and returns all related *endpoint references*. Basic operation that resolves a unique *logical name* to the corresponding *address(es)*. One *logical name* maps to at least one *endpoint reference*, but is unbound regarding the number of targets allowable. It is also possible that a given *endpoint reference* is referenced by more than one *logical name*.

Parameter: *QueryInput* (see 1.3.1.1)
Returns: *QueryResponse* (see 1.3.1.3)

The following parameter(s) MUST be specified in the parameterList of *QueryInput* (for values see 2.3.1):

QName	Description
LogicalReference	The *logical name* of the Logical Reference to resolve

The following properties MAY be specified in the propertyTypes of *QueryInput* (for values see 2.3.1):

QName	Description
EPR	*(returned by default, no need to specify in the propertyTypes list)*
Description	Description of the Logical Reference

2.3.2.5 updateEndpointReference

Updates all existing instances of the specified *endpoint reference*, affecting all Logical References referring to this *endpoint reference*.

Parameter: *ChangeInput* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)

The following properties MUST be specified in the parameterList of *ChangeInput*.
(for values see 2.3.1):

QName	Description
EPR	The value representing the Endpoint Reference to update

The following properties MUST be specified in the changeProperties of *ChangeInput*
(for values see 2.3.1):

QName	Description
EPR	The new Endpoint Reference value to be stored. This property value MUST be embedded in the *Update* change type element. (see section 1.3.1.2)

2.3.2.6 updateLogicalReference

Updates an existing *logical name* to *endpoint reference* mapping, enabling the caller to update the description of the Logical Reference and add and/or remove associated EPRs.

Parameter: *ChangeInput* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)

The following properties MUST be specified in the parameterList of *ChangeInput*
(for values see 2.3.1):

QName	Description
LogicalReference	The *logical name* of this Logical Reference

At least one property MUST be specified in the changeProperties of *ChangeInput*.
The following properties MAY be specified in the changeProperties of *ChangeInput*
(for values see 2.3.1):

QName	Description
Description	Description of the Logical Reference. This property value MUST be embedded in the *Update* change type element. (see section 1.3.1.2)
EPR	A single Endpoint Reference to be mapped or added to the mapping. This property value MUST be embedded in the *Update* change type element. (see section 1.3.1.2)

Note that more than one *EPR* element MAY be included in a single message exchange, effectively representing a list of values.

The *ChangeInput* parameter is fully capable of inserting, updating, and deleting properties in a single message exchange via the *changeProperties* component. This means that an *EPR* value may be used for adding a new EPR while another *EPR* value is sent identifying an existing *endpoint reference* that should be de-referenced. Values MUST be represented by the appropriate change type: Insert, Update, or Delete. (see section 1.3.1.2)

Appendix: RNS 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.ws.ibm.com"
    xmlns:tns="http://rns.ws.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" />

    <!-- Value Types -->
    <types>
      <xsd:schema targetNamespace="http://rns.ws.ibm.com"
                  xmlns:tns="http://rns.ws.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="ParameterList">
           <xsd:sequence>
             <xsd:any processContents="lax" minOccurs="1" maxOccurs="unbounded" />
           </xsd:sequence>
         </xsd:complexType>

            <xsd:complexType name="QueryInput">
              <xsd:sequence>
                    <!-- Dynamic list of parameters -->
                    <xsd:element ref="tns:parameterList"  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>
                    <!-- Dynamic list of parameters -->
                    <xsd:element ref="tns:parameterList"  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="QueryResponse">
                    <xsd:sequence>
                           <xsd:element ref="tns:baseDirectory" minOccurs="1"
maxOccurs="1"/>
                           <xsd:element ref="tns:endOfList"     minOccurs="1"
maxOccurs="1"/>
                           <xsd:any  minOccurs="0" maxOccurs="unbounded" />
                    </xsd:sequence>
            </xsd:complexType>

            <xsd:complexType name="ChangeResponse" />

            <!-- Parameter element declarations -->
            <xsd:element name="propertyTypes" type="xsd:QName"/>
            <xsd:element name="parameterList" type="tns:ParameterList"/>

            <!-- Resource property element declarations -->
            <xsd:element name="autoChangeDir"            type="xsd:boolean"/>
            <xsd:element name="autoResolve"              type="xsd:boolean"/>
            <xsd:element name="baseDirectory"            type="xsd:string"/>
            <xsd:element name="childCount"          type="xsd:int"/>
            <xsd:element name="directoryPath"            type="xsd:string"/>
            <xsd:element name="endOfList"                type="xsd:boolean"/>
            <xsd:element name="iteratorContextID"   type="xsd:string"/>
            <xsd:element name="iteratorIndex"            type="xsd:int"/>
            <xsd:element name="iteratorMaxAtOnce"   type="xsd:int"/>

            <!-- "Context" Resource for Maintaining State -->
            <xsd:element name="IteratorContext">
              <xsd:complexType>
                    <xsd:sequence>
                   <xsd:element ref="tns:childCount"        minOccurs="1" maxOccurs="1"/>
                   <xsd:element ref="tns:directoryPath"     minOccurs="1" maxOccurs="1"/>
                   <xsd:element ref="tns:iteratorContextID" minOccurs="1" maxOccurs="1"/>
```

```
                <xsd:element ref="tns:iteratorIndex"     minOccurs="0" maxOccurs="1"/>
                 </xsd:sequence>
              </xsd:complexType>
            </xsd:element>

  </xsd:schema>
</types>

<!-- RNS 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="ListResponseMessage">
       <part name="ListResponseMessage" type="tns:QueryResponse"/>
</message>
<message name="LookupInputMessage">
       <part name="LookupInputMessage" type="tns:QueryInput"/>
</message>
<message name="LookupResponseMessage">
       <part name="LookupResponseMessage" type="tns:QueryResponse"/>
</message>
<message name="UpdateInputMessage">
       <part name="UpdateInputMessage" type="tns:ChangeInput"/>
</message>
<message name="UpdateResponseMessage">
       <part name="UpdateResponseMessage" type="tns:ChangeResponse"/>
</message>
<message name="CreateInputMessage">
       <part name="CreateInputMessage" type="tns:ChangeInput"/>
</message>
<message name="CreateResponseMessage">
       <part name="CreateResponseMessage" type="tns:ChangeResponse"/>
</message>
<message name="DeleteInputMessage">
       <part name="DeleteInputMessage" type="tns:ChangeInput"/>
</message>
<message name="DeleteResponseMessage">
       <part name="DeleteResponseMessage" type="tns:ChangeResponse"/>
</message>

<!-- Adjunct Resource Properties Messages -->
<message name="DeletePropertyInputMessage">
       <part name="DeletePropertyInputMessage" type="tns:ChangeInput"/>
</message>
<message name="InsertPropertyInputMessage">
       <part name="InsertPropertyInputMessage" type="tns:ChangeInput"/>
</message>
<message name="ListPropertiesInputMessage">
       <part name="ListPropertiesInputMessage" type="tns:QueryInput"/>
</message>
<message name="UpdatePropertyInputMessage">
       <part name="UpdatePropertyInputMessage" type="tns:ChangeInput"/>
</message>

<!-- RRS Messages -->
<message name="ResolveInputMessage">
       <part name="ResolveInputMessage"          type="tns:QueryInput"/>
</message>
<message name="ResolveResponseMessage">
       <part name="ResolveResponseMessage"       type="tns:QueryResponse"/>
</message>
<message name="MapLogicalInputMessage">
       <part name="MapLogicalInputMessage"       type="tns:ChangeInput"/>
</message>
<message name="MapLogicalResponseMessage">
```

```
        <part name="MapLogicalResponseMessage"    type="tns:ChangeResponse"/>
</message>
<message name="CreateLogicalInputMessage">
        <part name="CreateLogicalInputMessage"    type="tns:ChangeInput"/>
</message>
<message name="CreateLogicalResponseMessage">
        <part name="CreateLogicalResponseMessage" type="tns:ChangeResponse"/>
</message>
<message name="DeleteLogicalInputMessage">
        <part name="DeleteLogicalInputMessage"    type="tns:ChangeInput"/>
</message>
<message name="DeleteLogicalResponseMessage">
        <part name="DeleteLogicalResponseMessage" type="tns:ChangeResponse"/>
</message>
<message name="UpdateLogicalInputMessage">
        <part name="UpdateLogicalInputMessage"    type="tns:ChangeInput"/>
</message>
<message name="UpdateLogicalResponseMessage">
        <part name="UpdateLogicalResponseMessage" type="tns:ChangeResponse"/>
</message>
<message name="DeleteEPRInputMessage">
        <part name="DeleteEPRInputMessage"    type="tns:ChangeInput"/>
</message>
<message name="DeleteEPRResponseMessage">
        <part name="DeleteEPRResponseMessage" type="tns:ChangeResponse"/>
</message>
<message name="UpdateEPRInputMessage">
        <part name="UpdateEPRInputMessage"    type="tns:ChangeInput"/>
</message>
<message name="UpdateEPRResponseMessage">
        <part name="UpdateEPRResponseMessage" type="tns:ChangeResponse"/>
</message>

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

        <!-- 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>

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

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

        <!-- Create Operation -->
        <operation name="create">
```

```
                <input  message="tns:CreateInputMessage"/>
                <output message="tns:CreateResponseMessage"/>
            </operation>

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

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

            <!-- Delete Adjunct Property Operation -->
            <operation name="deleteProperty">
                <input  message="tns:DeletePropertyInputMessage"/>
                <output message="tns:DeleteResponseMessage"/>
            </operation>

            <!-- Insert Adjunct Property Operation -->
            <operation name="insertProperty">
                <input  message="tns:InsertPropertyInputMessage"/>
                <output message="tns:CreateResponseMessage"/>
            </operation>

            <!-- List Adjunct Property Operation -->
            <operation name="listProperties">
                <input  message="tns:ListPropertiesInputMessage"/>
                <output message="tns:ListResponseMessage"/>
            </operation>

            <!-- Update Adjunct Property Operation -->
            <operation name="updateProperty">
                <input  message="tns:UpdatePropertyInputMessage"/>
                <output message="tns:UpdateResponseMessage"/>
            </operation>

    </portType>

    <!-- === Resource Resolution Service === -->
    <portType name="RNSResolverPortType">

            <!-- Logical Reference Resolve Operation -->
            <operation name="resolve">
                <input  message="tns:ResolveInputMessage"/>
                <output message="tns:ResolveResponseMessage"/>
            </operation>

            <!-- Logical Reference Create Operation -->
            <operation name="insertLogicalReference">
                <input  message="tns:CreateLogicalInputMessage"/>
                <output message="tns:CreateLogicalResponseMessage"/>
            </operation>

            <!-- Logical Reference Delete Operation -->
            <operation name="deleteLogicalReference">
                <input  message="tns:DeleteLogicalInputMessage"/>
                <output message="tns:DeleteLogicalResponseMessage"/>
            </operation>

            <!-- Logical Reference Update Operation -->
            <operation name="updateLogicalReference">
                <input  message="tns:UpdateLogicalInputMessage"/>
                <output message="tns:UpdateLogicalResponseMessage"/>
            </operation>

            <!-- Endpoint Reference Delete Operation -->
            <operation name="deleteEndpointReference">
```

```
                        <input  message="tns:DeleteEPRInputMessage"/>
                        <output message="tns:DeleteEPRResponseMessage"/>
                </operation>

                <!-- Endpoint Reference Update Operation -->
                <operation name="updateEndpointReference">
                        <input  message="tns:UpdateEPRInputMessage"/>
                        <output message="tns:UpdateEPRResponseMessage"/>
                </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

[5] OGSA Basic Profile 1.0 https://forge.gridforum.org/projects/ogsa-wg/document/draft-ggf-ogsa-basic-profile/en/

[6] XML Schema Part 2: Datatypes Second Edition http://www.w3.org/TR/xmlschema-2/