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

July 2005

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

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

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 limited 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.

The GFS-WG will propose a RNS profile for use with Grid File Systems. 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.

The reader is assumed to be familiar with WSRF and SOAP, see references [3] and [4] for background.

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.

1 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*, *logical 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 *logical 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 *logical 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 logical 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:

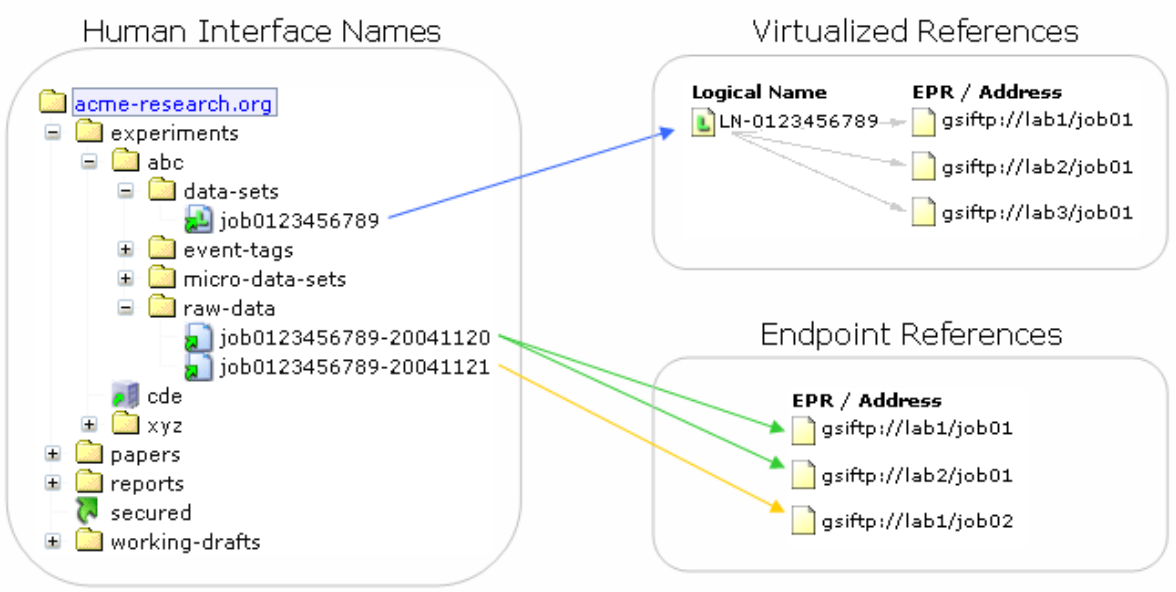

Figure 1 - Three-Tier Naming Architecture

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** or namespace *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:

QName [SOAP 1.2]	**Section**
Entry*	Entry Properties Message
Alias	1.1.2.4
Junction	1.1.2.1
LogicalReference	1.1.2.2
Referral	1.1.2.3
VirtualDirectory	1.1.1

* When the word "*entry*" is italic, this connotes reference to the basic namespace entry described above. The QName *Entry* is used in response messages to enclose the properties of an *entry* in the message structure.

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, it 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. Namespace hierarchies offer categorization or grouping of entries, 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. There are four basic types of junctions: *endpoint references*, *logical references, referrals*, and *aliases*.

1.1.2.1 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[2] Endpoint Reference (EPR) or URL. This is a many-to-many mapping, meaning that one namespace entry may reference many resources and one resource may be referenced by many namespace 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 = *Junction*

1.1.2.2 Logical Reference Junction

A *logical reference junction* is a junction that contains a context unique (potentially global) *logical name* and may contain either an endpoint reference (EPR) or universal resource locator (URL) that points to a secondary resolver service, like the companion *Resource Endpoint Resolution Service* (see section 3), for name-to-address resolution. Each *logical reference junction* MUST therefore embody one *logical name* and MAY identify any number of corresponding resolution contexts for the *logical name* to be resolved. The *logical name* value is retrieved using the *LogicalName* property, while the values of the resolver EPRs may be retrieved using the *LogicalResolvers* property. If a resolution context is not specified, the service SHOULD attempt to resolve the logical name using a default logical name resolution service, like the companion *Resource Endpoint Resolution Service* already noted (see section 3). This specification does not mandate a required format for the *logical name* nor the target resolver property value of a logical reference. In other words, the format of the *LogicalName* property is not mandated and neither is the format of the resolver EPRs.

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 section 3)

Corresponding resource property QName = *LogicalReference*

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/.

Since *referral junctions* point to other service instances, and the state and condition of these other service instances is not known, *referral junctions* may refer to unavailable or unresponsive service instances. To help guard against broken link conditions, multiple targets may be associated with a single *referral junction* to provide redundancy for greater availability.

For information regarding how the service handles *referral junctions*, please see Service Referrals in section 2.2.1.

Corresponding resource property QName = *Referral*

1.1.2.4 Alias Junction

An *alias junction* is a junction that references another entry within the same RNS service provider 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 conventional Unix filesystem hard links.

Support of *alias junctions* is **optional**. If an implementation chooses to support *alias junctions*, it SHOULD ensure referential validity for the lifetime of the alias. This means that the prescribed behaviour

of *alias junctions* within the RNS namespace SHOULD prevent and/or correct conditions that would orphan aliases, thereby yielding broken or invalid references.

Corresponding resource property QName = *Alias*

1.1.3 Pathnames

This specification relies heavily on the concept of paths and pathnames. In this context, a path is the route to a particular *entry* within the namespace, denoted by a string of characters signifying a series of names (representing namespace *entries*, most commonly virtual directories) that are separated by a delimiting character (the forward slash "/"). A pathname is the path of a namespace *entry* used as a, potentially global, unique identifier or "qualified name"; path and pathname may be used synonymously in this document.

1.1.3.1 Types of Pathnames

To qualify the scope of a path, thereby identifying its meaningful context, the following modifiers are used in this document: full, absolute, and global.

1.1.3.1.1 Full Path

A *full path* signifies the route to an *entry* from the root of the current namespace service instance. A *full path* MUST always be represented as a relative path from the root of the current service provider and therefore MUST NOT begin with a forward slash "/". *Full paths* MUST begin with a root-level *entry* name, always being relative from the root. An example of a *full path* may look like: `foo/bar/link1`

1.1.3.1.2 Absolute Path

An *absolute path* denotes a complete path starting from the root of the entire namespace. *Absolute paths* may span several namespace service providers or namespace service instances. An *absolute path* MUST ALWAYS begin with a forward slash "/". An example of an *absolute path* may look like: `/foo/bar/link1`

1.1.3.1.3 Global Path

A *global path* or *globally unique path* is simply an absolute path within a global namespace. A *global path* represents a globally unique namespace *entry* identifier, which ultimately symbolizes a globally unique resource name.

1.1.3.2 Entry Name Restrictions

Entry names are composed of a simple string of human readable characters. Since certain characters serve special purposes both within the namespace service and within a number of systems that may use this service, this section describes the mandatory restrictions for all *entry* names*:

Section	**Names MUST NOT…**
1.1.3.2.1	Contain any of the following characters: **\ / : ; * ? " < > |**
1.1.3.2.2	Contain any non-readable characters, such as the carriage return (ANSI 13) or line feed (ANSI 10) or tab (ANSI 9)
1.1.3.2.3	Be greater than 255 characters in length (Unicode)
	Names SHOULD…
1.1.3.2.4	Accommodate Unicode characters
1.1.3.2.5	Be easily readable by a human user, suggesting less than 32 characters per name
	Names MAY…
1.1.3.2.6	Contain space (ANSI 32) characters

* Notice these restrictions apply to *entry* names and are not describing paths. Paths are constructed of one or more *entry* names separated by the forward slash character (/). (see 1.1.3)

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[3] 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. For every new *list* request a new *IteratorContext* MUST be constructed 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.

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

The reader is assumed to be familiar with WSRF and SOAP, see references [3] and [4] for background.

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.[3].

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 instantiation of an *IteratorContext* resource is facilitated by the *createIteratorContext()* operation (see 1.3.4.3).

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="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>
```

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="iteratorContextID" 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 an instance of a valid namespace entry.)

QName	Description
Basic Properties	
AliasCount	Integer: Number of known aliases of *Entry.* The default value of this property is zero "0".
ChildCount	Integer: Number of subentries corresponding to *Entry*, if and only if *Entry* is a *VirtualDirectory*; 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: *Junction*, *LogicalReference*, *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*
LogicalName	String: Used to set , add, or retrieve the *logical name* of a *LogicalReference*
LogicalResolver	String: Used to set or add a single resolver of a *LogicalReference*
LogicalResolvers*	String: Used to retrieve all resolvers of a *LogicalReference*
Target*	String: Used to retrieve the target *entry* of an *Alias*.
TargetPath	String: Full path that identifies the target *entry* of an *Alias* junction. Since all *aliases* MUST be serviced by the same namespace service instance, this property is relative to the current working service instance (see 1.1.3.1.1)

* In a response message, these QNames are presented as SOAP message elements that hierarchically enclose child elements which represent complete embedded messages containing resource property values corresponding to the resource type denoted. For example, the *Target* property within a response message serves only to enclose an embedded message that contains all of the properties for the target *entry* of a given *Alias*. For an example of a response message that demonstrates this feature, see section 1.3.1.3 *QueryResponse*.

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. These properties are also referred to as *adjunct resource properties*. (See section 1.3.5)

1.2.2.2.3 Property Relationships

Property relationships describe how certain properties relate to one another. 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. Unlike programming objects, which may maintain accessors and mutators that enforce prescribed conditions between data members of an object, SOAP messages do not. Conditional relationships may be necessary, for instance, when preparing a message as an input parameter to an operation. The service MUST enforce such relationship requirements on the service side.

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
AutoResolve	Boolean value that if “true” will cause this operation to attempt to resolve any virtualized resources by their logical name using the companion Resource Endpoint Resolution 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**”]
BaseDirectory	String: Full path of the current working directory or *BaseDirectory*, relative to the current service instance. Construction of an absolute path may be realized by concatenating the *BaseDirectory* with an *entry* name. If an absolute path spans multiple service instances, the *BaseDirectory* value only reflects the path relative to the root of the current service instance. 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 returned in response to all query operations (see 1.3.1.3).
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.

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 properties or status of an *entry*.

To retrieve information about a particular namespace *entry*, a standard SOAP 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 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, in the Globus Toolkit 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 messages as an input parameter: *QueryInput* and *ChangeInput*, and returns one of the following messages respectively: *QueryResonse* and *ChangeResponse*.

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 [3]

```
<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. Only one change type message can be sent at a time.

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 four components: *BaseDirectory*, *EndOfList, ReferralEPR,* and an array of unrestrained message elements. If the response message signifies a referral message (see 2.2.1.1), then it will contain one or more referral EPRs. 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="0" maxOccurs="1"/>
    <xsd:element ref="tns:referralEPR"   minOccurs="0" maxOccurs="unbound"/>
    <xsd:any                             minOccurs="0" maxOccurs="unbound"/>
  </xsd:sequence>
</xsd:complexType>
```

An example *QueryResponse* message might look something like this, provided that the following properties were specified: *Name, ChildCount, Description, Type, EPRs, Target, TargetPath*

```
<ns1:BaseDirectory xmlns:ns1="http://rns.ggf.org"> ggf.org/ogsa/ </ns1:BaseDirectory>
<ns1:EndOfList xmlns:ns1="http://rns.ggf.org"> true </ns1:EndOfList>
<ns1:ReferralEPR xmlns:ns1="http://rns.ggf.org"> </ns1:ReferralEPR>
<ns1:Entry xmlns:ns1="http://rns.ggf.org">
  <ns1:Name> rns </ns1:Name>
  <ns1:ChildCount> 5 </ns1:ChildCount >
  <ns1:Description> This is the RNS project directory </ns1:Description>
  <ns1:Type> VirtualDirectory </ns1:Type>
</ns1:Entry>
<ns1:Entry xmlns:ns1="http://rns.ggf.org">
  <ns1:Name> some-resource </ns1:Name>
  <ns1:ChildCount> </ns1:ChildCount >
  <ns1:Description> This is a junction that points to some resource </ns1:Description>
  <ns1:Type> Junction </ns1:Type>
  <ns1:EPRs>
    <ns1:EPR> http://abc.com/some-resource </ns1:EPR>
    <ns1:EPR> http://xyz.com/some-resource </ns1:EPR>
  </ns1:EPRs>
</ns1:Entry>
<ns1:Entry xmlns:ns1="http://rns.ggf.org">
  <ns1:Name> rns-alias </ns1:Name>
  <ns1:Description> This is an alias to the RNS directory </ns1:Description>
  <ns1:Type> Alias </ns1:Type>
  <ns1:TargetPath> ggf.org/ogsa/rns </ns1:TargetPath>
  <ns1:Target>
    <ns1:Name> rns </ns1:Name>
    <ns1:ChildCount> 5 </ns1:ChildCount >
    <ns1:Description> This is the RNS project directory </ns1:Description>
    <ns1:Type> VirtualDirectory </ns1:Type>
  </ns1:Target>
</ns1:Entry>
```

1.3.1.4 ChangeResponse

This is a document literal service compliant message (complexType) that contains two components: *BaseDirectory* and a string message. If the response message signifies a referral message (see 2.2.1.1), then it will contain one or more referral EPRs. The following is the WSDL representation of the *ChangeResponse*:

```
<xsd:complexType name="ChangeResponse">
  <xsd:sequence>
    <xsd:element ref="tns:baseDirectory" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="tns:referralEPR"   minOccurs="0" maxOccurs="unbound"/>
    <xsd:element ref="tns:message"       minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

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).

If *Type* is *Alias*, then this operation MUST abort and return a fault message if the *TargetPath* value refers to any direct ancestor in the hierarchy, or refers to another *Alias* entry that ultimately refers to any direct ancestor in the hierarchy, or refers to an *entry* in another service instance. The first two conditions are intended to guard from circular referencing and since the *AliasCount* property and control is currently only possible within a single service instance, the last condition is necessary to ensure *Aliases* only reference other namespace *entries* within the same service instance.

This operation modifies namespace repository content and therefore SHOULD support update semantics that ensure atomic updates to namespace content. (See section 2.1 for details)

Parameter: *ChangeInput* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)
Faults: *RNSJunctionFault* (see 1.4.2), *RNSInvalidPropertyFault* (see 1.4.5), *RNSEntryExistsFault* (see 1.4.2), *RNSTypeFault* (see 1.4.7)

There are no changeProperties used in this operation.

Path MUST be specified in the parameterList of *ChangeInput*. If *Name* is specified, the service will implicitly append the value of *Name* to the value of *Path* to derive the full path of the entry to create; if *Name* is not specified, the service simply uses the value of *Path* as the full path of the entry to create.
(for values see 1.2.2.3):

QName	Description
Path	If *Name* is specified, this is the full path of the parent *virtual directory* of which *Name* will be created as a child entry; otherwise this is the full path of the *entry* to be created.
Name	String representation of the human interface name of the entry to create. **Optional:** If used denotes the name of the entry to be created as a child of *Path*; otherwise if not used, *Path* MUST represent the full path of the entry to create, including the entry name.
Type	String: Value denoting a type of *entry*; valid values are: *Junction*, *LogicalReference*, *Alias*, *Referral*, and *VirtualDirectory*

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

(continued on next page)

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

The following entry properties MAY be specified in the parameterList of *ChangeInput* (see 1.3.1.2). (for values see 1.2.2.2.1):

QName	Description
Description	String: Optional description
ModificationTime	DateTime (`xsd:dateTime`) representation of the last modified timestamp
EPR*	Value of a single Endpoint Reference to be associated with *Entry*
LogicalName	Value of the *logical name* to be associated with *Entry*, if *Entry* is a *LogicalReference*
LogicalResolver*	Value of a single *LogicalReference* to be associated with *Entry*
TargetPath	The full path of the target *entry*. Set only if *Entry* is an *Alias*.
Any adjunct resource property QNames and respective values set at runtime. See 1.3.5	

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

Properties within the parameter list of the *ChangeInput* message SHOULD correspond to the *Type* of *entry* created. If a property specified does not appropriately relate to the *Type* of *entry* designated, then a fault MAY be returned. For example, if the *Type* specified is *Junction* and a *TargetPath* or *LogicalName* property is also specified, an *RNSTypeFault* MAY be returned.

Example
The following example demonstrates how to create an Endpoint Reference Junction *entry* with two EPRs associated with it:

Abstract update *ChangeInput* message to create entry with two EPRs:

```
<ns1:ChangeInput xmlns:ns1="http://rns.ggf.org">
  <ns1:ParameterList xmlns:ns1="http://rns.ggf.org">
    <ns1:Path> /ggf.org/proj/rns/reports </ns1:Path>
    <ns1:Type> Junction </ns1:Type>
    <ns1:Description> This is a junction to RNS reports </ns1:Description>
    <ns1:EPR> gsiftp://xyz.com/rns/reports </ns1:EPR>
    <ns1:EPR> gsiftp://ns2.xyz.com/rns/reports </ns1:EPR>
  </ns1:ParameterList>
</ns1:ChangeInput>
```

1.3.2.2 delete

Enables an application to submit a request message that contains the path of the *entry* to delete. If the *entry* represents a *virtual directory*, then the designated *virtual directory* MUST NOT have any subentries associated with it; otherwise an *RNSDirectoryNotEmptyFault* MUST be returned.

This operation modifies namespace repository content and therefore SHOULD support update semantics that ensure atomic updates to namespace content. (See section 2.1 for details)

Parameter: *ChangeInput* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)
Faults: *RNSJunctionFault* (see 1.4.2), *RNSInvalidPropertyFault* (see 1.4.5), *RNSDirectoryNotEmptyFault* (see 1.4.2), *RNSEntryNotFoundFault* (see 1.4.4)

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

QName	Description
Path	The full 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, which is a subentry of the *virtual directory* entry denoted by the path value specified as an input parameter. If the path specified does not denote a *virtual directory* then a fault MAY be thrown.

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 the number of entries returned will be limited to this value. This feature is most useful when used in conjunction with an *IteratorContext*, which allows for sequential iteration through the result set of a list operation.

An *IteratorContext* is constructed and operated on using standard WSRF mechanisms for stateful resource interaction. In this case, the stateful resource is the point-in-time result set of a list operation. This result set embodies a "snapshot" of namespace subentries relative to the path specified, at the point in time in which the original *list()* operation was invoked. This ensures that if the namespace is modified between *list()* operations while using an *IteratorContext*, the list of subentries MUST remain coherent, corresponding to the result set derived at the time of the initial message exchange.

An *IteratorContext* MUST first be constructed, using the *createIteratorContext()* operation of the namespace port-type (see 1.3.4.3), followed by the use of the Endpoint Reference (EPR), derived from the *IteratorContextResponse*, to exchange messages with this *list()* operation. Once an *IteratorContext* is established and its corresponding EPR used for message exchanging, the *list()* operation MUST automatically maintain stateful result set information corresponding to the *IteratorContext* resource.

There are two basic methods of iterating through the point-in-time result set of the initial *list()* operation when using an *IteratorContext*:

1. **Implicit Iteration** – This method allows simple iteration in a convenient sequential fashion wherein the *IteratorIndex* advances implicitly. In this mode of operation, after the client or application exchanges an initial message with the *list()* operation using an *IteratorContext* EPR, thereby establishing a new *IteratorContext* resource, all subsequent messages to the *list()* operation using the *IteratorContext* EPR MUST return the next set of entries from the point-in-time result set. If a subsequent message is sent using a previously established *IteratorContext* EPR, with all parameters remaining the same, the service MUST automatically iterate progressively, according to the maximum number of subentries allowed per exchange, thereby responding with the maximum allowable number of entries starting at the first index that immediately follows the last interval and advancing the *IteratorIndex* value per operation message exchange.

   For example, if the *IteratorMaxAtOnce* property is set to a value of 5 and a new *IteratorContext* resource is constructed during an initial message to the *list()* operation, which yielded 10 entries in its point-in-time result set, then the iteration would look something like this:

```
[message][EPR]  -->  [list operation]  --> Result set compiled and
                <--  [entries 0,1,2,3,4]   state stored in
                                           IteratorContext resource
                                           with IteratorIndex = 5
[message][EPR]  -->  [list operation]
                <--  [entries 5,6,7,8,9 + EndOfList message]
[message][EPR]  -->  [destroy IteratorContext resource*]
```

(continued on next page)

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

Where [message] doesn't change and contains value like:

```
<Message>
  <wsa:EndpointReference>...</wsa:EndpointReference>
  <rns:Path> foo/bar </rns:Path>
  <rns:IteratorMaxAtOnce> 5 </rns:IteratorMaxAtOnce>
</Message>
```

*Since the IteratorContext resource is a WSRF resource, it is managed using standard WSRF mechanisms.

2. **Explicit Iteration** – This method offers more control over the iteration process by allowing the application or client to explicitly specify what segment of a given point-in-time result set should be returned. This is accomplished by specifying a base zero value for the *IteratorIndex* parameter. If a base zero value for the *IteratorIndex* parameter is specified, the service MUST set the internal list index marker, corresponding to the result set associated with the *IteratorContext* resource, to this value prior to responding to the request. The response MUST contain all entries from the specified *IteratorIndex* up to the index whose value is (*IteratorIndex* + *IteratorMaxAtOnce*) or up to the end of the list if the end of the list is reached prior to the value of (*IteratorIndex* + *IteratorMaxAtOnce*). As long as the *IteratorContext* resource is valid, the result set associated with the initial *list()* operation using the *IteratorContext* may be operated on, allowing overlapping or same segments to be returned in different message exchanges. Consequently, the application or client must guard from infinite loop conditions resulting from improper values specified for the *IteratorIndex*, such as specifying the same value repeatedly.

   For example, if the *IteratorMaxAtOnce* property is set to a value of 3 and 7 respectively and a new *IteratorContext* resource is constructed during an initial message to the *list()* operation, which yielded 10 entries in its point-in-time result set, then the iteration would look something like this:

```
[message1][EPR]  -->  [list operation]  --> Result set compiled and
                 <--  [entries 0,1,2]       state stored in
                                            IteratorContext resource
[message2][EPR]  -->  [list operation]
                 <--  [entries 3,4,5,6,7,8,9 + EndOfList message]
[message3][EPR]  -->  [destroy IteratorContext resource*]
```

   Where [message1] contains value like:

```
<Message>
  <wsa:EndpointReference>...</wsa:EndpointReference>
  <rns:Path>foo/bar</rns:Path>
  <rns:IteratorIndex>0</rns:IteratorIndex>
  <rns:IteratorMaxAtOnce>3</rns:IteratorMaxAtOnce>
</Message>
```

   Where [message2] contains value like:

```
<Message>
  <wsa:EndpointReference>...</wsa:EndpointReference>
  <rns:Path>foo/bar</rns:Path>
  <rns:IteratorIndex>3</rns:IteratorIndex>
  <rns:IteratorMaxAtOnce>7</rns:IteratorMaxAtOnce>
</Message>
```

   *Since the IteratorContext resource is a WSRF resource, it is managed using standard WSRF mechanisms.

(continued on next page)

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

Summary
In its simplest form, this operation returns a list of subentries relative to the path specified.

Large lists MAY be handled using an *IteratorContext* resource, via standard WSRF mechanisms, for stateful interaction with a point-in-time result set compiled during the initial *list()* operation request.

Automatic progressive sequential iteration MAY be realized by repeated message exchanges to the *list()* operation, provided a valid *IteratorContext* is used.

Specific iteration intervals and list segments MAY be rendered by setting the *IteratorIndex* parameter to a base zero value. The length of the segment returned is determined by the value of the *IteratorMaxAtOnce* parameter.

In all cases, if the end of the list is reached, the *EndOfList* element value is set to "true" and sent in the response message.

This operation only responds to requests whose path value denotes a namespace *virtual directory* (type: QName=*VirtualDirectory*). If the path specified does not denote a *virtual directory* then an *RNSTypeFault* SHOULD be returned.

Parameter: *QueryInput* (see 1.3.1.1)
Returns: *QueryResponse* (see 1.3.1.3)
Faults: *RNSJunctionFault* (see 1.4.2), *RNSInvalidPropertyFault* (see 1.4.5), *RNSEntryNotFoundFault* (see 1.4.4), *RNSTypeFault* (see 1.4.7)

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 full path of the *virtual directory* to list. **Required**.
AutoResolve	Boolean value that if "true" will cause this operation to (1) attempt to resolve any virtualized resources (one level) by their logical name using the companion Resource Endpoint Resolution Service and (2) resolve all *Alias TargetPath* values if an *Alias* points to another *Alias*, thereby returning the ultimate or "final destination" *TargetPath*. [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**"]
IteratorIndex	Integer indicating the (base zero) index of the result set corresponding with a valid *IteratorContext*. A positive value of this parameter MUST override the current list index marker in the *IteratorContext* maintained on the service end. A value greater than the total count of entries in the result set MAY throw a fault.

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

(continued on next page)

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

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 *VirtualDirectory*; 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: *Junction*, *LogicalReference*, *Alias*, *Referral*, and *VirtualDirectory*
EPRs	Used to retrieve all Endpoint References associated with *Entry*
LogicalName	Used to retrieve the *logical name* associated with *Entry*, if *Entry* is a *LogicalReference*
LogicalResolvers	Used to retrieve all resolvers corresponding to the *logical name* associated with *Entry*, if *Entry* is a *LogicalReference*
TargetPath	The full 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 1.3.5	

The following properties MAY be included in the *QueryResponse* message (see 1.3.1.3), this list includes all of the *Required Entry Properties* (see 1.2.2.2.1):

QName	AliasCount, ChildCount, Description, EPR, EPRs, LogicalName, LogicalResolver, LogicalResolvers, ModificationTime, Name, Target, TargetPath, Type

An example return message might look something like the following (if the `propertyTypes` of *QueryInput* included Name, ChildCount, Description, and Type):

```
<ns1:BaseDirectory xmlns:ns1="http://rns.ggf.org"> ggf.org/ogsa/ </ns1:BaseDirectory>
<ns1:EndOfList xmlns:ns1="http://rns.ggf.org"> true </ns1:EndOfList>
<ns1:Entry xmlns:ns1="http://rns.ggf.org">
  <ns1:Name>rns</ns1:Name>
  <ns1:ChildCount>5</ns1:ChildCount >
  <ns1:Description>This is the RNS project directory</ns1:Description>
  <ns1:Type>VirtualDirectory</ns1:Type>
</ns1:Entry>
<ns1:Entry xmlns:ns1="http://rns.ggf.org">
  <ns1:Name>byteio</ns1:Name>
  <ns1:ChildCount>3</ns1:ChildCount >
  <ns1:Description>This is the ByteIO project directory</ns1:Description>
  <ns1:Type>VirtualDirectory</ns1:Type>
</ns1:Entry>
```

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)
Faults: *RNSJunctionFault* (see 1.4.2), *RNSInvalidPropertyFault* (see 1.4.5), *RNSEntryNotFoundFault* (see 1.4.4)

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

QName	Description
Path	The full path of the *entry* to lookup. **Required**.
AutoResolve	Boolean value that if "true" will cause this operation to (1) attempt to resolve any virtualized resources (one level) by their logical name using the companion Resource Endpoint Resolution Service and (2) resolve all *Alias TargetPath* values if an *Alias* points to another *Alias*, thereby returning the ultimate or "final destination" *TargetPath*. [default value is "**false**"]

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 *Ent*
ChildCount	Number of subentries corresponding to *Entry*, if and only if *Entry* is a *VirtualDirectory*; 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: *Junction*, *LogicalReference*, *Alias*, *Referral*, and *VirtualDirectory*
EPRs	Used to retrieve all Endpoint References associated with *Entry*
LogicalName	String representation of the logical name of *Entry*, if *Entry* is a *LogicalReference*
LogicalResolvers	Used to retrieve all resolvers associated with the logical name of *Entry*
TargetPath	The full 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 1.3.5	

The following properties MAY be included in the *QueryResponse* message (see 1.3.1.3), this list includes all of the *Required Entry Properties* (see 1.2.2.2.1):

QName	AliasCount, ChildCount, Description, EPR, EPRs, LogicalName, LogicalResolver, LogicalResolvers, ModificationTime, Name, Target, TargetPath, Type

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.

This operation modifies namespace repository content and therefore SHOULD support update semantics that ensure atomic updates to namespace content. (See section 2.1 for details)

Parameter: *ChangeInput* (see 1.3.1.2)
Returns: *ChangeResponse* (see 1.3.1.4)
Faults: *RNSJunctionFault* (see 1.4.2), *RNSInvalidPropertyFault* (see 1.4.5), *RNSDirectoryNotEmptyFault* (see 1.4.2), *RNSEntryNotFoundFault* (see 1.4.4), *RNSEntryExistsFault* (see 1.4.2), *RNSTypeFault* (see 1.4.7)

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

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

The following entry properties MAY be specified in the changeProperties of *ChangeInput* in one of the following change types: *InsertType* (**I**), *UpdateType* (**U**), and *DeleteType* (**D**) (see 1.3.1.2).
(for values see 1.2.2.2.1):

QName	Description	I	U	D
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 full path the *entry* should be changed to. Used in "move" operations		✓	
EPR*	Add a single Endpoint Reference to be associated with *Entry*	✓		
EPRs**	Modify the entire list of Endpoint References associated with *Entry*		✓	✓
LogicalName	String representation of the logical name associated with *Entry*, if *Entry* is a *LogicalReference*		✓	
LogicalResolver*	Add a single resolver EPR if *Entry* is a *LogicalReference*	✓		
LogicalResolvers**	Modify the entire list of resolver EPRs associated with *Entry*, if *Entry* is a *LogicalReference*		✓	✓
TargetPath	Set the **full path** of the target entry. Set only if *Entry* is an *Alias*.		✓	
Type***	String: Value denoting a type of *entry*; valid values are: *Junction*, *LogicalReference*, *Alias*, *Referral*, and *VirtualDirectory*		✓	
Any adjunct resource property QNames and respective values set at runtime. See 1.3.5				

Each property MUST be enclosed in the appropriate change type message indicated by the **I**, **U**, and **D** columns. Only one change type message can be processed per request (see 1.3.1.2). If a property is enclosed in a change type message that is not marked with a check in the table above, the service SHOULD return a fault message. *(continued on next page)*

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

* Note that more than one *EPR* and *LogicalResolver* elements MAY be included in a single message exchange, effectively representing a list of values. Notice that a single *EPR* or *LogicalResolver* cannot be deleted or updated, since they are not identified by any corresponding handle or name. To remove or reorder these properties use the *UpdateType* message, which MUST reassign the comprehensive list of properties represented, ie. multiple EPRs.

** When these QNames are used the entire list will be affected. When using an *UpdateType* message, the service MUST reassign the list count and values to the count and values presented in the *UpdateType* message. For example, if an *entry* had the following EPRs assigned: A, B, and C and the user wanted to delete B, then the user would send an *UpdateType* message for QName "EPRs" that only contains A and C; this would effectively delete B. To delete all of the EPRs associated with an *entry*, simply use a *DeleteType* message for QName "EPRs".

*** If *Type* denotes a *VirtualDirectory*, indicating that the *Type* property will change, then the designated *virtual directory* MUST NOT have any subentries associated with it; otherwise an *RNSDirectoryNotEmptyFault* MUST be returned. Additionally, the *ChildCount* property MUST be nullified if changing from *Type=VirtualDirectory* to another. If *Type* denotes an *Alias*, the corresponding *TargetPath* property must be nullified when changed to another *Type*. If *Type* denotes a *Junction*, the corresponding *EPRs* properties must be nullified when changed to another *Type*. If *Type* denotes a *LogicalReference*, the corresponding *LogicalName* and *LogicalResolver* properties must be nullified when changed to another *Type*. If *Type* denotes a *Referral*, the corresponding *EPRs* properties must be nullified when changed to another *Type*.

When an *entry* is updated with a property type that is inconsistent with the *entry Type*, an *RNSTypeFault* SHOULD be returned. For example, if the *Type* is *VirtualDirectory* then inserting an *EPR* or *LogicalResolver* or *LogicalName* SHOULD cause an *RNSTypeFault.*

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

Example
The following example demonstrates how to add two EPRs to an existing Endpoint Reference Junction *entry*:

Entry before update:
```
<ns1:Entry xmlns:ns1="http://rns.ggf.org">
  <ns1:Name> reports </ns1:Name>
  <ns1:EPRs> <ns1:EPR>gsiftp://abc.com/rns/reports</ns1:EPR> </ns1:EPRs>
  <ns1:Type> Junction </ns1:Type>
</ns1:Entry>
```

Abstract update *ChangeInput* message to simply add two EPRs:
```
<ns1:ChangeInput xmlns:ns1="http://rns.ggf.org">
  <ns1:ParameterList xmlns:ns1="http://rns.ggf.org">
    <ns1:Path> /ggf.org/proj/rns/reports </ns1:Path>
  </ns1:ParameterList>
  <ns1:ChangeProperties wsrp:SetResourceProperties>
    <wsrp:Insert wsrp:Insert>
      <ns1:EPR> gsiftp://xyz.com/rns/reports </ns1:EPR>
      <ns1:EPR> gsiftp://ns2.xyz.com/rns/reports </ns1:EPR>
    </wsrp:Insert>
  </ns1:ChangeProperties>
</ns1:ChangeInput>
```

(continued on next page)

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

Entry after update:

```
<ns1:Entry xmlns:ns1="http://rns.ggf.org">
  <ns1:Name> reports </ns1:Name>
  <ns1:EPRs>
    <ns1:EPR>gsiftp://abc.com/rns/reports</ns1:EPR>
    <ns1:EPR>gsiftp://xyz.com/rns/reports</ns1:EPR>
    <ns1:EPR>gsiftp://ns2.xyz.com/rns/reports</ns1:EPR>
  </ns1:EPRs>
  <ns1:Type> Junction </ns1:Type>
</ns1:Entry>
```

Example Java snippet corresponding to the *ChangeInput* message listed above:

```
// Input values
String addr1 = "gsiftp://xyz.com/rns/reports";
String addr2 = "gsiftp://ns2.xyz.com/rns/reports";
String path  = "/ggf.org/proj/rns/reports";
// RNS, WSRF, and Axis Complex Types
ChangeInput      in = new ChangeInput();
InsertType   insert = new InsertType();
SetResourceProperties_Element srp = new SetResourceProperties_Element();
MessageElement[] me = new MessageElement[1];
// Set input parameter of parameterList
me[0] = (MessageElement)ObjectSerializer.toSOAPElement( path, PATH_QNAME );
in.setParameterList( new ParameterList( me ) );
// Set change parameter
me = new MessageElement[2];
me[0] = (MessageElement)ObjectSerializer.toSOAPElement( addr1, EPR_QNAME );
me[1] = (MessageElement)ObjectSerializer.toSOAPElement( addr2, EPR_QNAME );
insert.set_any( me );
srp.setInsert( insert );
in.setChangeProperties( srp );
// Submit request message to RNS service
rns.update( in );
```

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 full 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 full 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 full path of the *entry* to update. **Required**.

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 full 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 full path of the *virtual directory* to create.
Type	Set with a value of *VirtualDirectory*.

1.3.4 Iterator Context Operations

RNS facilitates the use of and interaction with a stateful resource, referred to as an *IteratorContext* (see 1.2.2.1), for the purpose of maintaining state information, such as the list index marker, while iterating large lists. In order to instantiate a new *IteratorContext* resource, an application or client must first send a message to the RNS service. The following operation provides this feature.

The RNS service MUST be able to support use of an *IteratorContext* and therefore MUST support the *createIteratorContext()* operation.

This is a document literal service compliant message (complexType) that contains one element:

Parameter Name	*Description*
iteratorContextID	String: This is an unconstrained string used to uniquely identify a particular IteratorContext instance. The format or syntax of this property is NOT mandated by this specification.

1.3.4.1 IteratorContextRequest

The following is the WSDL representation of the *IteratorContextRequest*:

```
<xsd:complexType name="IteratorContextRequest">
  <xsd:sequence>
    <!-- Iterator Context ID -->
    <xsd:element ref="tns:iteratorContextID" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="iteratorContextID" type="xsd:string" />
```

1.3.4.2 IteratorContextResponse

This is a document literal service compliant message (complexType) that contains two elements: an *Endpoint Reference* corresponding to the *IteratorContext* resource in context and the iterator context identification string.

The following is the WSDL representation of the *IteratorContextResponse*:

```
<xsd:complexType name="IteratorContextResponse">
  <xsd:sequence>
    <xsd:element ref="wsa:EndpointReference"/>
    <!-- Iterator Context ID -->
    <xsd:element ref="tns:iteratorContextID" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

1.3.4.3 createIteratorContext

Instantiate a new *IteratorContext*. This operation will create a new *IteratorContext*, using the provided *iteratorContextID* if specified, to be used in subsequent list operations. If an *iteratorContextID* is specified and already exists within the current instance of the service, then a fault message is returned. The response message of this operation embodies an Endpoint Reference that corresponds to the said service side *IteratorContext* resource.

Parameter: *IteratorContextRequest* (see 1.3.4.1)
Returns: *IteratorContextResponse* (see 1.3.4.2)

The following *iteratorContextID* MAY be specified; if *iteratorContextID* is not specified, then the service MUST generate a point-in-time unique identification string that signifies the *iteratorContextID*. If an *iteratorContextID* is generated by the service, the application or client can retrieve the *iteratorContextID* from the *IteratorContextResponse* message (see 1.3.4.2). The EPR contained in the response message MUST be used for all subsequent message exchanges that intend to interact with the *IteratorContext* resource.

QName	Description
IteratorContextID	Identification string of the *IteratorContext* to be created. **Optional.**

1.3.4.4 getIteratorContext

Open an existing *IteratorContext*. This operation will open the *IteratorContext* corresponding to the *IteratorContextID* specified. If a valid *IteratorContext* does not exists according to the *IteratorContextID* specified, then a fault message is returned. The response message of this operation embodies an Endpoint Reference that corresponds to the said service side *IteratorContext* resource.

Parameter: *IteratorContextRequest* (see 1.3.4.1)
Returns: *IteratorContextResponse* (see 1.3.4.2)

The following *iteratorContextID* MUST be specified. The EPR contained in the response message MUST be used for all subsequent message exchanges that intend to interact with the *IteratorContext* resource.

QName	Description
IteratorContextID	Identification string of the *IteratorContext* to be retrieved. **Required.**

1.3.5 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].

1.3.5.1 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:

1.3.5.2 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 1.3.5.1)::

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

1.3.5.3 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 1.3.5.1):

QName	Description
Name	Name of the *adjunct resource property* to be inserted. (QName)
DataType	WSDL compatible representation of the XML data type.

The following properties MAY be specified in the parameterList of *ChangeInput* (for values see 1.3.5.1):

QName	Description
Description	Description of the *adjunct resource property*
Profile	Optional string value denoting the Profile this *adjunct resource property* is associated with.

1.3.5.4 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 1.3.5.1):

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 1.3.5.1):

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 following properties MAY be included in the *QueryResponse* message (see 1.3.1.3), this list includes all of the *Profile Extension Operation Parameters* (see 1.3.5.1):

QName	DataType, Description, Name, Profile

1.3.5.5 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 1.3.5.1):

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

The following properties MAY be specified in the changeProperties of *ChangeInput*
(for values see 1.3.5.1):

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)

1.4 Operation Faults of the Resource Namespace Service

This section describes the use of faults in RNS. All RNS defined faults are based on Web Services standards, being fully compliant with WS-BaseFault[7]. This approach ensures all fault messages are constructed and handled in a common, standard complaint, way.

An RNS compliant implementation MUST employ all of the following faults:

1.4.1 RNSFault

This is the base fault defined by the RNS specification, providing a “superclass” for all other RNS faults.

Extends	*wsbf:BaseFaultType*
Element	**Description**
Path	String representation of the current working path where the service encountered the fault.

1.4.2 RNSDirectoryNotEmptyFault

This fault MUST be returned when a *delete* or *update* operation is targeted at a *virtual directory* that has subentries associated with it.

Extends	*RNSFault*

1.4.3 RNSEntryExistsFault

This fault MUST be returned when a *create* or *update* operation attempts to create an *entry* that already exists in the namespace or attempts to rename/move an *entry* to a path that corresponds to an existing *entry*.

Extends	*RNSFault*

1.4.4 RNSEntryNotFoundFault

This fault MUST be returned when any operation fails to resolve a path. This fault is only applicable to paths that correspond to the context of the current service instance namespace. Please see section 2.2.1 for more information regarding connecting multiple RNS namespaces and the use of service referrals to redirect applications and clients to other service instances that may precede or succeed the current service instance in terms of path name hierarchy.

Extends	*RNSFault*

1.4.5 RNSInvalidPropertyFault

This fault MUST be returned when any operation is issued a QName as a resource property name that is invalid. In addition to the predefined resource properties (1.2.2.2.1) RNS facilitates the ability to dynamically create user defined resource properties (1.2.2.2.2); this fault SHOULD only be returned if the property name specified does not exist in both the predefined and the user defined resource property lists.

Extends	*RNSFault*

Element	**Description**
PropertyName	The invalid property QName that was specified.

1.4.6 RNSJunctionFault

This fault MUST be returned when any operation is issued a path that traverses beyond a *Junction entry* in the RNS namespace. As the service attempts to resolve an absolute path, and encounters a *Junction entry* before the path is fully resolved, then a fault message containing all of the EPRs associated with the junction encountered MUST be returned. This fault is only applicable to paths that traverse *Junction entries*, not *referrals*. Please see section 2.2.1 for more information regarding connecting multiple RNS namespaces and the use of service referrals. Regardless of the *Type* of *Junction entry*, this fault MUST always contain all of the EPRs associated; if the junction is a *LogicalReference*, for example, the service MUST first attempt to resolve the logical name to its respective endpoint reference(s).

Extends	*RNSFault*
Element	**Description**
EPRs	All EPRs registered with the *Junction entry* that was traversed during path resolution.

1.4.7 RNSTypeFault

This fault SHOULD be returned when a *list* operation specifies a path that resolves to an *entry* with a *Type* value not equal to *VirtualDirectory*. This fault SHOULD be returned when a *create* operation specifies a properties for an *entry* that are invalid for that *entry Type*.

Extends	*RNSFault*

2 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 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.

2.1 Distributed Namespace Repositories

A namespace service that accommodates scalability, redundancy/availability, and geographic dissemination implicitly necessitates the distribution of servers in a grid or network. Duplicate or replica copies of namespace content, which embody namespace entries and their associated properties, MAY need to be distributed within a network and therefore the specification of the namespace service MUST mandate provisions to make such configurations possible.

Namespace content is persistently stored in what is referred to in this document as a *repository*. Each RNS service provider provides services for a namespace derived by the names and associated mappings that are contained within the corresponding repository. This specification does not prescribe the type of data store to be used as the repository of namespace content. Furthermore, implementation specific arrangements, such as sharing a single repository between multiple RNS service providers or enable a single RNS service provider to leverage multiple repositories, are not mandated by this specification.

This specification does not mandate how namespace content is to be distributed. It does not specify how to distribute, replicate, or maintain consistency between multiple repository replicas. However, since the specification does describe operations that allow for namespace content to be updated, it must therefore require that all updated operations comply with the following update semantics. Any operation that modifies namespace repository content SHOULD support update semantics that ensure atomic updates to namespace content. This means that operations like *create*, *delete*, and *update* SHOULD guarantee synchronized processing that prevents update contingencies based on concurrent execution.

2.2 Resolution Spanning Namespace Services

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 repository boundaries. There are two fundamental approaches to resolving path names that span multiple namespace domains or service instances: *service referrals* and *delegated resolution*.

2.2.1 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 an 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.

This specification mandates that a compliant implementation MUST facilitate the capability of service referrals as described here. In particular, the mechanism MUST use the *ReferralEPR* element featured in both the *QueryResponse* and *ChangeResponse* messages.

2.2.1.1 Referral Messages

Since all namespace entities are uniquely distinguished by a globally unique pathname, and pathnames are the only option for identifying namespace entries, this specification REQUIRES that all service operations accommodate the use of absolute pathnames. However, since RNS namespace service providers MAY be distributed and arranged as a hierarchy of namespace services, secondary and tertiary service providers will represent subsections of the total namespace, each with respectively different root directories. If absolute pathnames span multiple service providers then the service must use a mechanism to interrupt the operation with a referral message that redirects the client or application to the appropriate service instance of the next repository; provided that the path can be resolved by the current operating RNS service provider and a referral junction is encountered while attempting to resolve the path.

In the case where the beginning of a path specified cannot be resolved, and the RNS service provider is a secondary or tertiary instance, the service MUST either (1) return a referral message referencing its parent service provider with a *BaseDirectory* value of "/" or (2) return an *RNSEntryNotFoundFault* message (see section 1.4.4). To reference a parent or superior (upper level) service provider requires that the implementation is aware of, and can produce a valid *ReferralEPR* to, the parent service provider. This is an implementation specific configuration option that is not mandated by this specification.

Each referral message MUST contain a *BaseDirectory* value (see 1.2.2.3), which is a full path of the current working directory (see 1.1.3.1.1), relative to the current service provider. Notice that the *BaseDirectory* does not denote the global path from the perspective of the client or application (see 1.1.3.1.3), but rather the full path relative to the root of the current service provider.

All namespace requests are responded to using a *QueryResponse* or *ChangeResponse* message. Both of these message types define a *ReferralEPR* element, which is ONLY used when the message denotes a referral message. If a request is made using a path that traverses a referral namespace entry, the request MUST be interrupted with a referral message. A referral message is therefore defined as a response message that contains one or more referral EPRs. If a *QueryResponse* or *ChangeResponse* message defines a non-null value for the *ReferralEPR* element, the application or client MUST redirect the request to the service provider specified by the referral EPR. The application or client is responsible for handling all details related to connecting to multiple service providers and maintaining a mapping between each provider's *BaseDirectory* and the absolute path.

The following illustration provides a very basic example of how a referral message is used:

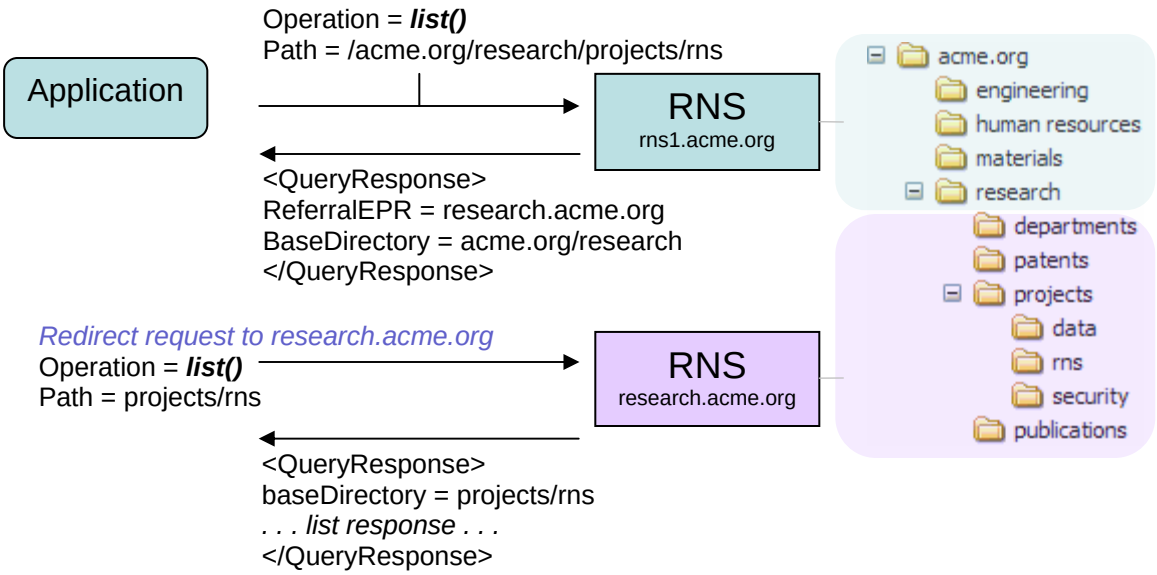

Figure 2

A good understanding regarding the use of absolute paths and the *BaseDirectory* value of each response message is essential to the understanding of spanning multiple service providers and handling referral messages. The values of a *QueryResponse* or *ChangeResponse* message are relative to the context of the service provider that retuned it. For example, in Figure 2, the *BaseDirectory* value of "acme.org/research" is the full path relative to the service provider "rns1.acme.org". Secondly, the value of *ReferralEPR*, which is "research.acme.org", represents the secondary service provider to redirect the request to.

Since the *BaseDirectory* value in a referral message is the full path relative to the service provider that returned it, it signifies the portion of the absolute pathname that should be factored out of the pathname to be used in the redirected operation. In the example presented in Figure 2, the *BaseDirectory* value of "acme.org/research" is factored out of the absolute path "/acme.org/research/projects/rns" to yield the pathname "projects/rns" to be used in the redirected operation. Notice that if the new pathname spans yet another repository boundary and therefore redirected to another service, this already factored path must undergo refactoring to be used within the context of the next service provider that is serving the third repository.

Notice that each user or application initiated operation MAY use an absolute path since the provider specific pathnames are automatically derived by the service when necessary. Optionally, a client implementation MAY choose to retain connection information per service provider and therefore incorporate the necessary functionality to derive the appropriate context specific full path that corresponds to the connection serving that previously visited portion of the namespace.

2.2.2 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 quite 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.

3 Resource Endpoint Resolution Service

The Resource Endpoint 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 *logical 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*.

3.1 RNS Resolver Basic Components

RNS Resolver is comprised of two fundamental service components: *logical names* and *endpoint references* or *addresses*. These two basic components when coupled together are referred to as *logical references* and 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:

3.1.1 Logical Reference

A Logical Reference (*LogicalReference*) 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.

3.1.2 Endpoint Reference

An Endpoint Reference (*EPR*) 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[2]

3.2 Document Style Messaging

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

3.3 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 *logical references*.

To retrieve information about a particular *logical 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.

3.3.1 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*
LogicalName	Used to set or update the *logical name*

3.3.2 RNS Resolver Operations

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

3.3.2.1 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 3.3.1):

QName	Description
EPR	The Endpoint Reference to be deleted

3.3.2.2 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 3.3.1):

QName	Description
LogicalName	The *logical name* to delete

3.3.2.3 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 3.3.1):

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

The following properties MAY be specified in the parameterList of *ChangeInput* (for values see 3.3.1):

QName	Description
Description	Description of the *LogicalReference*

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

3.3.2.4 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 3.3.1):

QName	Description
LogicalName	The *logical name* to resolve

The following properties MAY be specified in the propertyTypes of *QueryInput* (for values see 3.3.1):

QName	Description
EPRs	*(returned by default, no need to specify in the propertyTypes list)*
Description	Description of the *LogicalReference*

The following properties MAY be included in the *QueryResponse* message (see 1.3.1.3), this list includes some of the *Resolver Operation Parameters* (see 3.3.1):

QName	Description, EPRs, EPR

3.3.2.5 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 3.3.1):

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 3.3.1):

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)

3.3.2.6 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 3.3.1):

QName	Description
LogicalName	The *logical name* denoting the *LogicalReference* to update

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 3.3.1):

QName	Description
Description	Description of the *LogicalReference*. 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)

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 area of interest and will require further investigation.
- Backup – Backup of RNS data may be required.
- Replication – Replication of namespace repository data is not discussed however may be necessary for redundancy in high availability deployments.
- Repository Consistency – As a namespace service that is intended to enable federation of multiple, hierarchical, namespace services, coherency of namespace repository data between distributed service instance datastores is essential. This specification does not mandate where nor how namespace repository data is stored. Therefore, coherency between redundant and delegated RNS services is not addressed.
- Discussion of access control lists (ACLs) within RNS, their purpose, scope, representation, and enforcement. There are two fundamental levels of consideration, (1) access control to namespace information and (2) access control to the target resource that the namespace refers to. The latter case most often is protected independent of the namespace referring to it.

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. It also purposed a set of operations needed to provide a companion service to the basic RNS namespace services for resolving logical, or *abstract*, names; the Resource Endpoint Resolution Service (RNS Resolver). 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.

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

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.ggf.org"
    xmlns:tns="http://rns.ggf.org"
    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.ggf.org"
       xmlns:tns="http://rns.ggf.org"
       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 === -->

        <!-- "IteratorContextRequest" Object for Creating/Opening IteratorContexts -->
        <xsd:element name="IteratorContextRequest">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="tns:iteratorContextID" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
```

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

<xsd:element name="IteratorContextResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wsa:EndpointReference"/>
      <xsd:element ref="tns:iteratorContextID" minOccurs="1" maxOccurs="1"/>
    </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="0" maxOccurs="1"/>
    <xsd:element ref="tns:referralEPR"   minOccurs="0" maxOccurs="unbound"/>
    <xsd:any  minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ChangeResponse">
  <xsd:sequence>
    <xsd:element ref="tns:baseDirectory" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="tns:referralEPR"   minOccurs="0" maxOccurs="unbound"/>
    <xsd:element ref="tns:message"       minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<!-- RNS Fault declarations -->
<xsd:complexType name="RNSFaultType">
  <xsd:complexContent>
    <xsd:extension base="wsbf:BaseFaultType">
      <xsd:sequence>
        <xsd:element name="path" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="RNSDirectoryNotEmptyFaultType">
  <xsd:complexContent>
    <xsd:extension base="tns:RNSFaultType"/>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="RNSEntryExistsFaultType">
  <xsd:complexContent>
    <xsd:extension base="tns:RNSFaultType"/>
```

```
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="RNSEntryNotFoundFaultType">
    <xsd:complexContent>
      <xsd:extension base="tns:RNSFaultType"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="RNSInvalidPropertyFaultType">
    <xsd:complexContent>
      <xsd:extension base="tns:RNSFaultType">
        <xsd:sequence>
          <xsd:element name="propertyName" type="xsd:string"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="RNSJunctionFaultType">
    <xsd:complexContent>
      <xsd:extension base="tns:RNSFaultType">
        <xsd:sequence>
          <xsd:element name="EPRs" type="xsd:string"
                       minOccurs="1" maxOccurs="unbound"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="RNSTypeFaultType">
    <xsd:complexContent>
      <xsd:extension base="tns:RNSFaultType"/>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="RNSFault" type="tns:RNSFaultType"/>
  <xsd:element name="RNSDirectoryNotEmptyFault"
               type="tns:RNSDirectoryNotEmptyFaultType"/>
  <xsd:element name="RNSEntryExistsFault" type="tns:RNSEntryExistsFaultType"/>
  <xsd:element name="RNSEntryNotFoundFault" type="tns:RNSEntryNotFoundFaultType"/>
  <xsd:element name="RNSInvalidPropertyFault" type="tns:RNSInvalidPropertyFaultType"/>
  <xsd:element name="RNSJunctionFault" type="tns:RNSJunctionFaultType"/>
  <xsd:element name="RNSTypeFault" type="tns:RNSTypeFaultType"/>

  <!-- 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"/>
  <xsd:element name="message"             type="xsd:string"/>
  <xsd:element name="referralEPR"         type="xsd:string"/>

  <!-- "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="IteratorContextRequest">
  <part name="IteratorContextRequest" element="tns:IteratorContextRequest"/>
</message>
<message name="IteratorContextResponse">
  <part name="IteratorContextResponse" element="tns:IteratorContextResponse"/>
</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>


<!-- WSDL messages for each distinct fault -->
<wsdl:message name="RNSFaultMessage">
  <wsdl:part name="fault" element="tns:RNSFault"/>
</wsdl:message>
<wsdl:message name="RNSDirectoryNotEmptyFaultMessage">
  <wsdl:part name="fault" element="tns:RNSDirectoryNotEmptyFault"/>
</wsdl:message>
<wsdl:message name="RNSEntryExistsFaultMessage">
  <wsdl:part name="fault" element="tns:RNSEntryExistsFault"/>
</wsdl:message>
<wsdl:message name="RNSEntryNotFoundFaultMessage">
  <wsdl:part name="fault" element="tns:RNSEntryNotFoundFault"/>
</wsdl:message>
<wsdl:message name="RNSInvalidPropertyFaultMessage">
  <wsdl:part name="fault" element="tns:RNSInvalidPropertyFault"/>
</wsdl:message>
<wsdl:message name="RNSJunctionFaultMessage">
  <wsdl:part name="fault" element="tns:RNSJunctionFault"/>
</wsdl:message>
<wsdl:message name="RNSTypeFaultMessage">
  <wsdl:part name="fault" element="tns:RNSTypeFault"/>
</wsdl:message>

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

  <!-- Operation for creating a new IteratorContext -->
  <operation name="createIteratorContext">
    <input  message="tns:IteratorContextRequest"/>
    <output message="tns:IteratorContextResponse"/>
  </operation>

  <!-- Operation for retrieving an existing IteratorContext -->
  <operation name="getIteratorContext">
    <input  message="tns:IteratorContextRequest"/>
    <output message="tns:IteratorContextResponse"/>
  </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>

  <!-- Create Operation -->
  <operation name="create">
    <input  message="tns:CreateInputMessage"/>
    <output message="tns:CreateResponseMessage"/>
    <fault  name="RNSJunctionFault"
            message="tns:RNSJunctionFaultMessage"/>
    <fault  name="RNSEntryExistsFault"
            message="tns:RNSEntryExistsFaultMessage"/>
    <fault  name="RNSInvalidPropertyFault"
            message="tns:RNSInvalidPropertyFaultMessage"/>
    <fault  name="RNSTypeFault"
            message="tns:RNSTypeFaultMessage"/>
    <fault  name="RNSFault"
            message="tns:RNSFaultMessage"/>
  </operation>

  <!-- Delete Operation -->
  <operation name="delete">
    <input  message="tns:DeleteInputMessage"/>
    <output message="tns:DeleteResponseMessage"/>
    <fault  name="RNSJunctionFault"
            message="tns:RNSJunctionFaultMessage"/>
    <fault  name="RNSInvalidPropertyFault"
            message="tns:RNSInvalidPropertyFaultMessage"/>
    <fault  name="RNSDirectoryNotEmptyFault"
            message="tns:RNSDirectoryNotEmptyFaultMessage"/>
    <fault  name="RNSEntryNotFoundFault"
            message="tns:RNSEntryNotFoundFaultMessage"/>
    <fault  name="RNSFault"
            message="tns:RNSFaultMessage"/>
  </operation>

  <!-- List Operation -->
  <operation name="list">
    <input  message="tns:ListInputMessage"/>
    <output message="tns:ListResponseMessage"/>
    <fault  name="RNSJunctionFault"
            message="tns:RNSJunctionFaultMessage"/>
    <fault  name="RNSInvalidPropertyFault"
            message="tns:RNSInvalidPropertyFaultMessage"/>
    <fault  name="RNSEntryNotFoundFault"
            message="tns:RNSEntryNotFoundFaultMessage"/>
    <fault  name="RNSTypeFault"
            message="tns:RNSTypeFaultMessage"/>
    <fault  name="RNSFault"
            message="tns:RNSFaultMessage"/>
  </operation>

  <!-- Lookup Operation -->
  <operation name="lookup">
    <input  message="tns:LookupInputMessage"/>
    <output message="tns:LookupResponseMessage"/>
    <fault  name="RNSJunctionFault"
            message="tns:RNSJunctionFaultMessage"/>
    <fault  name="RNSInvalidPropertyFault"
            message="tns:RNSInvalidPropertyFaultMessage"/>
    <fault  name="RNSEntryNotFoundFault"
            message="tns:RNSEntryNotFoundFaultMessage"/>
    <fault  name="RNSFault"
            message="tns:RNSFaultMessage"/>
  </operation>

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

```
      <output message="tns:UpdateResponseMessage"/>
      <fault  name="RNSJunctionFault"
              message="tns:RNSJunctionFaultMessage"/>
      <fault  name="RNSInvalidPropertyFault"
              message="tns:RNSInvalidPropertyFaultMessage"/>
      <fault  name="RNSDirectoryNotEmptyFault"
              message="tns:RNSDirectoryNotEmptyFaultMessage"/>
      <fault  name="RNSEntryNotFoundFault"
              message="tns:RNSEntryNotFoundFaultMessage"/>
      <fault  name="RNSEntryExistsFault"
              message="tns:RNSEntryExistsFaultMessage"/>
      <fault  name="RNSTypeFault"
              message="tns:RNSTypeFaultMessage"/>
      <fault  name="RNSFault"
              message="tns:RNSFaultMessage"/>
    </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 Endpoint 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>
```

Acknowledgements

Noriyuki Soda (SRA)
Takuya Ishibashi (SOUM)

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

case the procedures for copyrights defined in the GGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the GGF or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE GLOBAL GRID FORUM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."

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] Web Services Addressing 1.0 – Core (W3C Working Draft 31 March 2005)
http://www.w3.org/TR/ws-addr-core/

[3] Web Services Resource Properties (WS-ResourceProperties) Version 1.2 06/10/2004
http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf

[4] SOAP Version 1.2 Part 1: Messaging Framework
http://www.w3.org/TR/soap12-part1/

[5] OGSA Basic Profile 1.0
https://forge.gridforum.org/projects/ogsa-wg/document/draft-ggf-ogsa-wsrf-basic-profile/en/20

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

[7] (WS-BaseFaults) Web Services Base Faults 1.2 (Working Draft 02, June 24, 2004)
http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-02.pdf

[WSDL] Web Services Description Language (WSDL) 1.1
http://www.w3.org/TR/wsdl

[WS-BaseNotification 1.2] Web Service Base Notification 1.2 (Working Draft 03, 21 June 2004)
http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf