GFD-R-P.101
GFS-WG

Manuel Pereira, IBM Almaden Research Center
Osamu Tatebe, University of Tsukuba
Leo Luan, IBM Almaden Research Center
Ted Anderson, IBM Almaden Research Center

September 22, 2006
[Revised: March 1, 2007]

Resource Namespace Service Specification

Status of This Document

This document provides information to the Grid community about resource namespace services. Distribution is unlimited.

Copyright Notice

Abstract

This document describes the specification of the Resource Namespace Service (RNS), which offers a simple standard way of mapping names to endpoints within a grid or distributed network. It functions as a Web Services registry that provides name-to-resource mapping. Names are mapped to endpoint references and are stored in a persistent namespace repository. RNS offers hierarchical rendering of human-oriented names, which makes resource mappings easily accessible and usable by human interface applications. 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 describing the port type, functions, and features of the Resource Namespace Service.

Contents

Manuel Pereira, IBM Almaden Research Center
Osamu Tatebe, University of Tsukuba
Leo Luan, IBM Almaden Research Center
Ted Anderson, IBM Almaden Research Center

1.1 Introduction

The Resource Namespace Service, which will henceforth be referred to as 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. It enables construction of a uniform, global, hierarchical namespace [1]. In concept, RNS supports a three-tier naming architecture, which consists of *human interface names*, *logical references*, and *endpoint references* (see “Three-tier Naming” in the considerations section).

RNS is intended to facilitate namespace, or registry, 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 reader is assumed to be familiar with Web Services in general, document style messaging, and SOAP [4].

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.

When describing abstract data models, this specification uses the notational convention used by the XML Infoset [XML Infoset]. Specifically, abstract property names always appear in square brackets (e.g., [some property]).

When describing concrete XML schemas [XML Schema Part 1, Part 2], this specification uses the notational convention of WS-Security [WS-Security]. Specifically, each member of an element’s [children] or [attributes] property is described using an XPath-like notation (e.g., /x:MyHeader/x:SomeProperty/@value1). The use of {any} indicates the presence of an element wildcard (<xs:any/>). The use of @{any} indicates the presence of an attribute wildcard (<xs:anyAttribute/>).

XML Namespaces

This specification uses a number of namespace prefixes throughout; they are listed in Table 1. Note that the choice of any namespace prefix is arbitrary and not semantically significant (see [XML Namespaces]).

Prefix	Namespace
s11	http://schemas.xmlsoap.org/soap/envelope
Xsd	http://www.w3.org/2001/XMLSchema
Wsa	http://www.w3.org/2005/03/addressing
Rns	http://schemas.ogf.org/rns/2006/09/rns

1.2 Basic Namespace Components

All strings indexed and used as human oriented names within RNS are referred to as namespace entries. There are two fundamental types of entries described in this document, they are referred to as *virtual directories* and *junctions*. These two basic entry types are further described as follows:

1.2.1 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 namespace service client, a virtual directory functions similar to that of a standard filesystem directory or registry key. It is considered *virtual* because it does not have any corresponding representation outside of the namespace. A virtual directory, therefore, is purely a namespace entity that functions in much the same way as a conventional filesystem directory or registry key by maintaining a list of subentries, which thereby demonstrate a hierarchical relationship. There are no restrictions regarding the layout of the namespace tree; both virtual directories and junctions can be nested within nested virtual directories recursively.

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

1.2.2 Junctions

A *junction* is an RNS entry that interconnects a reference to an existing resource into the hierarchical namespace. Junctions represent a name-to-resource mapping that is composed of a human oriented index key or “name” that maps to an endpoint reference. The endpoint reference may refer to any addressable resource, which includes other namespace entries, as well as names or unique identifiers to be resolved by other resolution service, as well as definitive target consumable resource. All compliant RNS implementations MUST embody the target information of a namespace junction within a valid WS-Addressing [2] Endpoint Reference (EPR).

1.2.2.1 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*:

Names MUST NOT…	
1.2.2.1.1	Contain any of the following characters: `\ / : ; * ? ‟ < > |`
1.2.2.1.2	Contain any non-readable characters, such as the carriage return (ANSI 13) or line feed (ANSI 10) or tab (ANSI 9)
1.2.2.1.3	Be greater than 255 characters in length (Unicode)
Names SHOULD…	
1.2.2.1.4	Accommodate Unicode characters
1.2.2.1.5	Be easily readable by a human user, suggesting less than 32 characters per name
Names MAY…	
1.2.2.1.6	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 “Pathnames” in the Considerations section).

1.3 Resource Namespace Service Port Type

The Resource Namespace Service port type enables Web Services access to named resources arranged in a hierarchical manifestation. As a result, the RNS port type is composed of simple namespace entry operations that enable the construction of and access to a hierarchical mapping of named resources.

The RNS port type embodies the following operations:

```
      add( string: entry_name, EndpointReferenceType: entry_reference, ... )

      list( string: entry_name_regexp )

      update( EndpointReferenceType: parent, string: entry_name,
EndpointReferenceType: entry_reference, ... )

      query()

      remove( string: entry_name_regexp )
```

1.3.1 Namespace Operations

The following is a comprehensive list of operations that MUST be implemented to be compliant with the RNS namespace port type (RNSPortType) specification.

1.3.1.1 add

This operation is used to define a new name-to-resource mapping, or a virtual directory, to be created and persistently stored by the service host. The name provided will be mapped to the endpoint reference specified and stored as a sub-entry within the directory this operation is executed against. The directory will henceforth be referred as an operating virtual directory.

If the entry name specified matches an existing entry name already registered as a sub-entry of the current operating virtual directory, then an *RNSEntryExistsFault* MUST be returned. Equality matching SHOULD be case specific, in other words the names should be found to match exactly with an existing entry before a fault is returned. If this operation is executed against a namespace junction, then an RNSEntryNotDirectoryFault MUST be returned.

If the value of the entry reference is empty then a virtual directory entry will be created instead of a junction.

This operation modifies namespace repository content and therefore SHOULD support update semantics that ensure atomic updates to namespace content.

```
<rns:add>
  <rns:entry_name> xsd:string </rns:entry_name>
  {any}*
  <rns:entry_reference>
      wsa:EndpointReferenceType
  </rns:entry_reference>
</rns:add>
```

The components of the `add` request message are further described as follows:

`/rns:add/entry_name`
This name to be used as the human interface name for this namespace entry. This name will be displayed, as specified, as a sub-entry within the operating directory.

`/rns:add/entry_reference`
The WS-Addressing EndpointReferenceType to be registered as the corresponding reference of the name specified. This effectively represents the resource that the name is mapping to. This resource might be another namespace entry, a logical abstract identifier to be resolved by another service, or an endpoint reference to a consumable resource.*

* RNS does not distinguish between types of reference used.

The response to the `add` request message, all of whose components were successfully processed, MUST be a message of the following form:

```
<rns:addResponse>
  <rns:entry_reference>
      wsa:EndpointReferenceType
  </rns:entry_reference>
</rns:addResponse>
```

The component of the `addResponse` response message is further described as follows:

`/rns:addResponse/entry_reference`

A WS-Addressing EndpointReferenceType that refers to the newly added namespace entry.

1.3.1.1.1 Example SOAP Encoding of the add Message Exchange

Below is a simple example of how to register a namespace entry named “foo”—that maps to an endpoint reference that refers to some service or resource—within directory “A”.

The following is a non-normative example of an `add` message using SOAP 1.1:

```
<s11:Envelope
      xmlns:s11=”http://www.w3.org/2003/05/soap-envelope”
      xmlns:wsa=”http://www.w3.org/2005/03/addressing”
      xmlns:rns=”http://schemas.ogf.org/rns/2006/09/rns”>
    <s11:Header>
      <wsa:Action>
        http://schemas.ogf.org/rns/2006/09/rns/add
      </wsa:Action>
      <wsa:To s11:mustUnderstand=”1”>
        http://abc.com/rns/A
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:add>
        <rns:entry_name> foo </rns:entry_name>
        <rns:entry_reference>
            <wsa:Address> http://xyz.com/misc/foo </wsa:Address>
        </rns:entry_reference>
      </rns:add>
    </s11:Body>
</s11:Envelope>
```

The following is a non-normative example of an `addResponse` message using SOAP 1.1:

```
<s11:Envelope
      xmlns:s11=”http://www.w3.org/2003/05/soap-envelope”
      xmlns:wsa=”http://www.w3.org/2005/03/addressing”
      xmlns:rns=”http://schemas.ogf.org/rns/2006/09/rns”>
    <s11:Header>
      <wsa:Action>
        http://schemas.ogf.org/rns/2006/09/rns/addResponse
      </wsa:Action>
      <wsa:To>
        http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:addResponse>
        <rns:entry_reference>
            <wsa:Address> http://abc.com/rns/A/foo </wsa:Address>
        </rns:entry_reference>
      </rns:addResponse>
    </s11:Body>
</s11:Envelope>
```

1.3.1.2 list

This operation is used to render a list of sub-entries associated with the current operating directory. A subset of entries may be returned when a regular expression pattern is used to identify the entries to be included in the subset. Notice that the list returned strictly represents the respective list of entries, registered as sub-entries (or child entries) of the current operating directory, at the point in time in which the request message was received. Notice that the list does not recursively return entries even when a sub-entry is a virtual directory.

A compliant service implementation MUST be capable of handling an optional string argument (`rns:entry_name_regexp`) used to accommodate *regular expression* statements intended to identify a subset of sub-entries to be listed. This specification does not specify any particular *regular expression* notation or syntax.

If this operation is executed against a namespace junction, then an *RNSEntryNotDirectoryFault* MUST be returned.

```
<rns:list>
  <rns:entry_name_regexp> xsd:string </rns:entry_name_regexp>
</rns:list>
```

The components of the `list` request message are further described as follows:

`/rns:list/entry_name_regexp`
: An optional string used to identify a subset of entries to be listed. The value of this string MAY be empty, in which case a comprehensive listing of the sub-entries is returned; otherwise the value of this string is intended to represent a regular expression statement that distinguishes a particular subset of entries to be returned.

The response to the `list` request message, all of whose components were successfully processed, MUST be a message of the following form:

```
<rns:listResponse>
  rns:EntryList
</rns:listResponse>
```

The component of the `listResponse` response message is further described as follows:

`/rns:EntryList`
: This is a “complex type” message that embodies the entire entry list, including associated entry attributes. Details of the `rns:EntryList` message are described below.

1.3.1.2.1 EntryList Complex Type

```
<rns:EntryList>
  <rns:entry>
    <rns:entry_name> xsd:string </rns:entry_name>
    {any}*
    <rns:entry_reference>
      wsa:EndpointReferenceType
    </rns:entry_reference>
  </rns:entry>*
</rns:EntryList>
```

The components of the `rns:EntryList` message are further described as follows:

`/rns:EntryList/entry`
: A single entry enclosing all associated attributes of the entry within sub-elements.

`/rns:EntryList/entry/entry_name`
: The name of the entry represented.

`/rns:EntryList/entry/entry_reference`
: The WS-Addressing EndpointReferenceType of the entry represented; this is the endpoint reference that is mapped to the entry name.

1.3.1.2.2 Example SOAP Encoding of the list Message Exchange

Below is a simple example of how to list all sub-entries of the current operating directory “A”.

The following is a non-normative example of a `list` message using SOAP 1.1:

```
<s11:Envelope
      xmlns:s11=”http://www.w3.org/2003/05/soap-envelope”
      xmlns:wsa=”http://www.w3.org/2005/03/addressing”
      xmlns:rns=”http://schemas.ogf.org/rns/2006/09/rns”>
    <s11:Header>
      <wsa:Action>
        http://schemas.ogf.org/rns/2006/09/rns/list
      </wsa:Action>
      <wsa:To s11:mustUnderstand=”1”>
        http://abc.com/rns/A
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:list>
        <rns:entry_name_regexp> </rns:entry_name_regexp>
      </rns:list>
    </s11:Body>
</s11:Envelope>
```

The following is a non-normative example of a `listResponse` message using SOAP 1.1:

```
<s11:Envelope
      xmlns:s11=”http://www.w3.org/2003/05/soap-envelope”
      xmlns:wsa=”http://www.w3.org/2005/03/addressing”
      xmlns:rns=”http://schemas.ogf.org/rns/2006/09/rns”>
    <s11:Header>
      <wsa:Action>
        http://schemas.ogf.org/rns/2006/09/rns/listResponse
      </wsa:Action>
```

```
       <wsa:To>
         http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
       </wsa:To>
     </s11:Header>

     <s11:Body>
       <rns:listResponse>
         <rns:EntryList>
           <rns:entry>
             <rns:entry_name> foo </rns:entry_name>
             <rns:entry_reference>
               <wsa:Address> http://xyz.com/misc/foo </wsa:Address>
             </rns:entry_reference>
           </rns:entry>
           <rns:entry>
             <rns:entry_name> bar </rns:entry_name>
             <rns:entry_reference>
               <wsa:Address> http://123.com/bar </wsa:Address>
             </rns:entry_reference>
           </rns:entry>
         </rns:EntryList>
       </rns:listResponse>
     </s11:Body>
</s11:Envelope>
```

1.3.1.3 update

This operation is used to move or rename an existing entry, and update the entry reference of an existing entry. The object of this operation is the current service endpoint, which may be a virtual directory or a namespace junction. The parameters used in this operation reflect destination values for the current entry and a new entry reference.

The destination can be specified by an entry name and the parent directory. If the parent directory specified is not a virtual directory, then an *RNSEntryNotDirectoryFault* MUST be returned. If the parent directory specified is not managed by the same RNS service of the current service endpoint, then an *RNSCrossServiceFault* MUST be returned. If the parent directory specified is one of its own subdirectories, then an RNSEntryInvalidFault MUST be returned.

If the object of this operation is a virtual directory, the entry reference specified SHOULD be empty. If it is not empty, then an RNSEntryIsDirectoryFault MUST be returned.

This operation modifies namespace repository content and therefore SHOULD support update semantics that ensure atomic updates to namespace content.

```
<rns:update>
  <rns:entry_parent>
      wsa:EndpointReferenceType
  </rns:entry_parent>
  <rns:entry_name> xsd:string </rns:entry_name>
  {any}*
  <rns:entry_reference>
     wsa:EndpointReferenceType
  </rns:entry_reference>
</rns:update>
```

The components of the `update` request message are further described as follows:

`/rns:update/entry_parent`
The WS-Addressing EndpointReferenceType that identifies the virtual directory to which this entry should be moved. If the value of this element refers to the current parent directory, or if no value is specified, the hierarchical position of this entry within the namespace will remain unchanged.

`/rns:update/entry_name`
This element identifies the value of the name this entry should be renamed to. If the value of this element reflects the current name of the entry, or if no value is specified, the name of this entry will remain unchanged.

`/rns:update/entry_reference`
The WS-Addressing EndpointReferenceType to be newly registered. If the value of this element reflects the current entry reference of the entry, or if no value is specified, the entry reference of this entry will remain unchanged.

Notice that at least one of the elements should be specified for any action to take place. Optionally three elements may be specified to accomplish both a hierarchical move and an entry rename as well as an entry reference update within a single transaction.

The response to the `update` request message, all of whose components were successfully processed, MUST be a message of the following form:

```
<rns:updateResponse>
  <rns:entry_reference>
      wsa:EndpointReferenceType
  </rns:entry_reference>
</rns:updateResponse>
```

The component of the `updateResponse` response message is further described as follows:

`/rns:updateResponse/entry_reference`
A WS-Addressing EndpointReferenceType that refers to the newly moved/renamed/updated namespace entry.

1.3.1.3.1 Example SOAP Encoding of the update Message Exchange

Below is a simple example of how to rename a namespace entry from “foo” to “bar”.

The following is a non-normative example of an `update` message using SOAP 1.1:

```
<s11:Envelope
      xmlns:s11=”http://www.w3.org/2003/05/soap-envelope”
      xmlns:wsa=”http://www.w3.org/2005/03/addressing”
      xmlns:rns=”http://schemas.ogf.org/rns/2006/09/rns”>
    <s11:Header>
      <wsa:Action>
        http://schemas.ogf.org/rns/2006/09/rns/update
      </wsa:Action>
      <wsa:To s11:mustUnderstand=”1”>
        http://abc.com/rns/A/foo
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:update>
        <rns:entry_name> bar </rns:entry_name>
      </rns:update>
    </s11:Body>
</s11:Envelope>
```

The following is a non-normative example of an `updateResponse` message using SOAP 1.1:

```
<s11:Envelope
      xmlns:s11=”http://www.w3.org/2003/05/soap-envelope”
      xmlns:wsa=”http://www.w3.org/2005/03/addressing”
      xmlns:rns=”http://schemas.ogf.org/rns/2006/09/rns”>
    <s11:Header>
      <wsa:Action>
        http://schemas.ogf.org/rns/2006/09/rns/updateResponse
      </wsa:Action>
      <wsa:To>
        http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:updateResponse>
        <rns:entry_reference>
            <wsa:Address> http://abc.com/rns/A/bar </wsa:Address>
        </rns:entry_reference>
      </rns:updateResponse>
    </s11:Body>
</s11:Envelope>
```

1.3.1.4 query

This operation is used to query, or get the properties of, an existing name-to-resource mapping. This operation simply returns all of the registered properties associated with a namespace entry (name-to-resource mapping), including an endpoint reference to the parent directory entry. This operation differs from the list operation in that the list operation is executed against a virtual directory entry for the purpose of retrieving a list of sub-entries and their associated properties, whereas this operation is executed against any namespace entry for the purpose of retrieving all associated properties and an endpoint reference to the parent directory of the entry. Retrieving an endpoint reference to the parent directory is particularly useful when traversing the namespace hierarchy upward (often referred to as “backward lookup”).

If this operation is executed against a root node in the namespace hierarchy, then an endpoint reference to itself SHOULD be returned.

```
<rns:query />
```

There are no argument elements specified by this operation.

The response to the `query` request message MUST be a message of the following form:

```
<rns:queryResponse>
  rns:EntryProperties
</rns:queryResponse>
```

The component of the `queryResponse` response message is further described as follows:

`/rns:EntryProperties`

This is a “complex type” message that embodies all of the properties associated with the entry, including an endpoint reference to the virtual directory that is represented as the parent directory in the namespace hierarchy. Details of the `rns:EntryProperties` message are described below.

1.3.1.4.1 EntryProperties Complex Type

```
<rns:EntryProperties>
  <rns:entry_parent>
      wsa:EndpointReferenceType
  </rns:entry_parent>
  <rns:entry_name> xsd:string </rns:entry_name>
  {any}*
  <rns:entry_reference>
      wsa:EndpointReferenceType
  </rns:entry_reference>
</rns:EntryProperties>
```

The components of the `rns:EntryProperties` message are further described as follows:

`/rns:EntryProperties/entry_parent`
The WS-Addressing EndpointReferenceType referring to the parent virtual directory entry of the entry represented.

`/rns:EntryProperties/entry_name`
The name of the entry represented.

`/rns:EntryProperties/entry_reference`
The WS-Addressing EndpointReferenceType of the entry represented; this is the endpoint reference that is mapped to the entry name.

1.3.1.4.2 Example SOAP Encoding of the query Message Exchange

Below is a simple example of how to query a namespace entry named “foo”.

The following is a non-normative example of a `query` message using SOAP 1.1:

```
<s11:Envelope
      xmlns:s11=”http://www.w3.org/2003/05/soap-envelope”
      xmlns:wsa=”http://www.w3.org/2005/03/addressing”
      xmlns:rns=”http://schemas.ogf.org/rns/2006/09/rns”>
    <s11:Header>
      <wsa:Action>
        http://schemas.ogf.org/rns/2006/09/rns/query
      </wsa:Action>
      <wsa:To s11:mustUnderstand=”1”>
        http://abc.com/rns/A
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:query />
    </s11:Body>
</s11:Envelope>
```

The following is a non-normative example of a `queryResponse` message using SOAP 1.1:

```
<s11:Envelope
      xmlns:s11="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://www.w3.org/2005/03/addressing"
      xmlns:rns="http://schemas.ogf.org/rns/2006/09/rns">
    <s11:Header>
      <wsa:Action>
        http://schemas.ogf.org/rns/2006/09/rns/queryResponse
      </wsa:Action>
      <wsa:To>
        http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:queryResponse>
        <rns:EntryProperties>
          <rns:entry_parent>
            <wsa:Address> http://abc.com/rns/A </wsa:Address>
          </rns:entry_parent>
          <rns:entry_name> foo </rns:entry_name>
          <rns:entry_reference>
            <wsa:Address> http://xyz.com/misc/foo </wsa:Address>
          </rns:entry_reference>
        </rns:EntryProperties>
      </rns:queryResponse>
    </s11:Body>
</s11:Envelope>
```

1.3.1.5 remove

This operation is used to remove, or "unlink", an existing name-to-resource mapping. A compliant service implementation MUST be able to perform string equality comparisons to determine if the string specified represents a sub-entry to be removed. A compliant service implementation MAY also implement *regular expression* capabilities to enable the identification of multiple sub-entries in a single remove request message.

If this operation is executed against a namespace junction, then an *RNSEntryNotDirectoryFault* MUST be returned. If the sub-entry specified 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.

```
<rns:remove>
  <rns:entry_name> xsd:string </rns:entry_name>
</rns:remove>
```

The components of the `remove` request message are further described as follows:

`/rns:remove/entry_name`

The name of the sub-entry to be removed from the operating directory. This operation will dissolve the association of the name-to-resource mapping, therefore removing both entry name and entry reference from the service's persistent data store.

The value of this string MAY embody regular expression for the purpose of identifying more than one sub-entry.

The response to the `remove` request message, all of whose components were successfully processed, MUST be a message of the following form:

```
<rns:removeResponse>
  rns:EntryList
</rns:removeResponse>
```

The component of the `removeResponse` response message is further described as follows:

`/rns:EntryList`

This is a "complex type" message that embodies the entire list of entries that were successfully removed, including respective entry attributes. Details of the `rns:EntryList` message are described in section 1.3.1.2.1.

1.3.1.5.1 Example SOAP Encoding of the remove Message Exchange

Below is a simple example of how to remove a namespace entry from "foo".

The following is a non-normative example of a `remove` message using SOAP 1.1:

```
<s11:Envelope
      xmlns:s11="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://www.w3.org/2005/03/addressing"
      xmlns:rns="http://schemas.ogf.org/rns/2006/09/rns">
    <s11:Header>
      <wsa:Action>
        http://schemas.ogf.org/rns/2006/09/rns/remove
      </wsa:Action>
      <wsa:To s11:mustUnderstand="1">
        http://abc.com/rns/A
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:remove>
        <rns:entry_name> foo </rns:entry_name>
      </rns:remove>
    </s11:Body>
</s11:Envelope>
```

The following is a non-normative example of a `removeResponse` message using SOAP 1.1:

```
<s11:Envelope
      xmlns:s11="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://www.w3.org/2005/03/addressing"
      xmlns:rns="http://schemas.ogf.org/rns/2006/09/rns">
    <s11:Header>
      <wsa:Action>
        http://schemas.ogf.org/rns/2006/09/rns/removeResponse
      </wsa:Action>
      <wsa:To>
        http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
      </wsa:To>
    </s11:Header>
    <s11:Body>
      <rns:removeResponse>
        <rns:EntryList>
          <rns:entry>
            <rns:entry_name> foo </rns:entry_name>
            <rns:entry_reference>
              <wsa:Address> http://xyz.com/misc/foo </wsa:Address>
            </rns:entry_reference>
          </rns:entry>
        </rns:EntryList>
      </rns:removeResponse>
    </s11:Body>
</s11:Envelope>
```

1.4 Operation Faults

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 *remove* operation is executed against a virtual directory that has sub-entries associated with it.

Extends	*RNSFault*

1.4.3 RNSEntryExistsFault

This fault MUST be returned when an *add* operation attempts to create an *entry* that already exists within the same virtual directory.

Extends	*RNSFault*

1.4.4 RNSEntryNotDirectoryFault

This fault MUST be returned when an *add* operation, a *list* operation or a *remove* operation is executed against an entry that is not a directory, or when the destination directory specified in an *update* operation is not a directory.

Extends	*RNSFault*

1.4.5 RNSCrossServiceFault

This fault MUST be returned when an *update* operation attempts to move to an entry that is not managed by the same RNS service of the current service endpoint.

Extends	*RNSFault*

1.4.6 RNSEntryInvalidFault
This fault MUST be returned when the destination directory specified in an *update* operation is one of its own subdirectories managed by the same RNS service of the current service endpoint.

Extends	*RNSFault*

1.4.7 RNSEntryIsDirectoryFault
This fault MUST be returned when the entry reference is specified against a virtual directory in an *update* operation.

Extends	*RNSFault*

1.5 Considerations
This section includes several significant points to consider when composing a profile designed to render RNS for a specific application or purpose. These considerations are also pertinent to implementers of such namespace services, offering a small list of factors that might prove to be critical to security, performance, scalability, usability, and the overall efficiency of a namespace service.

1.5.1 Security
The topic of security as a whole is not discussed in this specification document. Nevertheless, security remains crucial to the protection of sensitive information. Information integrity and privacy are included in this consideration. Security is recognized as a substantial necessity in nearly all OGSA services. RNS is intended to function with independent security solutions made available by the Web Services and OGSA communities.

1.5.1.1 Access Control Lists
Although this document does not discuss access control lists (ACLs), there has been noteworthy discussion regarding, their purpose, scope, representation, and enforcement in RNS. 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.

1.5.2 Extensible Resource Properties
The ability to manage the number and type of resource properties, or namespace entry attribute fields, in a dynamic run-time fashion would offer a more efficient and flexible way to take an RNS compliant implementation and deploy it in various environments, potentially with disparate profiles applied, all without changing the run-time implementation.

1.5.3 Three-tier Naming
RNS is intended to operate in harmony with other resolution services to facilitate a comprehensive naming and registry service that is composed of three-tiers. A three-tiered naming architecture supports 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 namespace 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/abstract references or unique identifiers), which in turn map logical references 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.

1.5.4 Unification of Distributed Resource Namespace Services

RNS is intended to be capable of facilitating widely distributed namespaces, with the ultimate capability of a global namespace. 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.

1.5.4.1 Distributed of 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. Though this specification does mandate such provisions, it does not mandate where or how namespace repository data is stored. Therefore, coherency between redundant and delegated RNS services remains a detail that should be addressed at some level, for example within a profile rendering document or an implementation design document.

1.5.4.2 Pathnames

In the context of federating RNS instances, the concept of paths and pathnames are relied upon heavily. 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) 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.

1.5.4.3 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.

1.5.5 Root-level Names and Resolution Services

To realize a globally scalable, universal and federated namespace, conventions for root-level resolution service providers may need to be established. Root-level names and resolution services may be considered analogous to the management, governing authority, and distribution and support of "top-level domains" and root servers in DNS. The implication of configuring and utilizing root level RNS services includes at least the following considerations:

- Governing authority of root level names – There may need to be leveraged/established an organization that will take responsibility for root-level names, for example ICANN [8].
- Root-level RNS service providers – Currently in the IP realm, DNS servers are most typically distributed and maintained by several volunteer organizations.
- Discovery and designation – Applications that utilize the RNS namespace may need to incorporate some type of preliminary mechanism to identify what root-level RNS service provider it should communicate with, this could be as simple as a configuration file and as sophisticated as UDDI.

1.5.6 Iterators for Large Directories

Since RNS is intended to facilitate namespace and registry services for a wide variety of service consumers, many of which may potentially store information in very large directories or registry keys; this presents a significant performance impediment. In an effort to reduce this obstruction, we suggest using an interactive, stateful, iterator that is capable of rendering segments of a complete list of entries. Ideally, the iterator should be capable of maintaining a position marker that is capable of advancing forward and backward within the result set list.

1.5.7 Backup and Namespace Data Management

Backup and management of RNS data is recommended but is not described in this document. Backup and management details will ultimately be implementation and potentially deployment specific.

Acknowledgements

Takuya Ishibashi (SOUM)

Author Information

Manuel Pereira[1], Leo Luan, Ted Anderson
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120, USA
manuel@trinitybiblechurch.org
leoluan@us.ibm.com
ota@us.ibm.com

Osamu Tatebe
Department of Computer Science, University of Tsukuba
1-1-1 Tennodai, Tsukuba
Ibaraki 3058573 Japan
tatebe@cs.tsukuba.ac.jp

Intellectual Property Statement

Disclaimer

[1] This author's work related to this document was conducted during his prior employment with IBM.

Full Copyright Notice

References

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

[2] 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] Simple Object Access Protocol (SOAP) 1.1 (W3C Note 08 May 2000)
http://www.w3.org/TR/2000/NOTE-SOAP-20000508

[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

[8] Internet Corporation For Assigned Names and Numbers
http://www.icann.org/

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

[XML Namespaces]
W3C Recommendation "Namespaces in XML", Tim Bray, Dave Hollander, Andrew Layman, 14 January, 1999
http://www.w3.org/TR/1999/REC-xml-names-19990114/

[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