Network Working Group
Request for Comments: 3367
Category: Standards Track

N. Popp
M. Mealling
VeriSign, Inc.
M. Moseley
Netword, Inc.

August 2002

Common Name Resolution Protocol (CNRP)

Status of this Memo

Copyright Notice

Abstract

People often refer to things in the real world by a common name or phrase, e.g., a trade name, company name, or a book title. These names are sometimes easier for people to remember and type than URLs. Furthermore, because of the limited syntax of URLs, companies and individuals are finding that the ones that might be most reasonable for their resources are being used elsewhere and so are unavailable. For the purposes of this document, a "common name" is a word or a phrase, without imposed syntactic structure, that may be associated with a resource.

This effort is about the creation of a protocol for client applications to communicate with common name resolution services, as exemplified in both the browser enhancement and search site paradigms. Although the protocol's primary function is resolution, it is also intended to address issues of internationalization and localization. Name resolution services are not generic search services and thus do not need to provide complex Boolean query, relevance ranking or similar capabilities. The protocol is a simple, minimal interoperable core. Mechanisms for extension are provided, so that additional capabilities can be added.

Table of Contents

1. Introduction

Services are arising that offer a mapping from common names to Internet resources (e.g., as identified by a URI). These services often resolve common name categories such as company names, trade names, or common keywords. Thus, such a resolution service may operate in one or a small number of categories or domains, or may expect the client to limit the resolution scope to a limited number of categories or domains. For example, the phrase "Internet Engineering Task Force" is a common name in the "organization" category, as is "Moby Dick" in the book category.

Two classes of clients of such services are being built, browser improvements and web accessible front-end services. Browser enhancements modify the "open" or "address" field of a browser so that a common name can be entered instead of a URL. Internet search sites integrate common name resolution services as a complement to search. In both cases, these may be clients of back-end resolution services. In the browser case, the browser must talk to a service that will resolve the common name. The search sites are accessed via a browser. In some cases, the search site may also be the back-end resolution service, but in others, the search site is a front-end to a collection of back-end services.

This effort is about the creation of a protocol for client applications to communicate with common name resolution services, as exemplified in both the browser enhancement and search site paradigms. Name resolution services are not generic search services and thus do not need to provide complex Boolean query, relevance ranking or similar capabilities. The protocol is a simple, minimal interoperable core. Mechanisms for extension are provided, so that additional capabilities can be added.

Several other issues, while of importance to the deployment of common name resolution services, are outside of the resolution protocol itself and are not in the initial scope of the proposed effort. These include discovery and selection of resolution service providers, administration of resolution services, name registration, name ownership, and methods for creating, identifying or insuring unique common names.

For the purposes of this document, a "common name" is a word or a phrase, without imposed syntactic structure, that may be associated with a resource. These common names will be used primarily by humans, as opposed to machine agents. A common name "resolution service" handles these associations between common names and data (resources, information about resources, pointers to locations, etc.). A single common name may be associated with different data records, and more than one resolution service is expected to exist. Any common name may be used in any resolution service.

Common names are not URIs (Uniform Resource Identifiers) in that they lack the syntactic structure imposed by URIs; furthermore, unlike URNs, there is no requirement of uniqueness or persistence of the association between a common name and a resource. (Note: common names may be expressed in a URI, the syntax for which is described in RFC 3368 [9].)

This document will define a protocol for the parameterized resolution necessary to make common names useful. "Resolution" is defined as the retrieval of data associated (a priori) with descriptors that match the input request. "Parameterized" means the ability to have a multi-property descriptor. Descriptors are not required to provide unique identification, therefore 0 or more records may be returned to meet a specific input query.

2. Important Notes

2.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [7].

2.2 DTD is Definitive

The descriptive portions of this document contain pieces of XML that are *illustrative examples only*. Section 5 of this document contains the XML DTD for CNRP, which is definitive. If any discrepancies are found, the DTD wins.

2.3 Uniform Resource Identifiers

All URIs used within the CNRP protocol MUST adhere to the 'absoluteURI' production found in the ABNF of [3]. CNRP does not define the semantics of a Base and therefore is not capable of expressing the 'URI-Reference' production.

3. Interaction Model

3.1 Services, Servers, Datasets and Referrals

CNRP assumes a particular interaction model where a generalized "service" provides common name resolution at one or more actual "servers". If the data contained in all its servers is identical (mirrors), the service need not identify any particular subset of data. If, however, the service provides different collections of data through different servers (e.g., subsets, specialized collections, etc.), it SHOULD indicate what subsets of its data that each server offers. This is done by using URIs to uniquely disambiguate one dataset from another. If the service offers a copy of a collection of data on agreement with a foreign service, the foreign service SHOULD provide a dataset URI to allow the collection to be identified as related to its own offerings.

CNRP supports the concept of referrals. This is where a server can know that another Service exists, within the same Service or elsewhere, that can provide further answers to a particular query but decides to forward that fact onto the client instead of chaining the query for the client. A referral is sent along with the rest of the results from a server (if any). Referrals to a service SHOULD indicate the particular dataseturi that triggered the referral, if it is known. See Section 4.2.5 for details on referrals and loop detection.

3.2 Requests and Responses

The protocol consists of a simple request/response mechanism. A client sends one of a few types of requests to a server which responds with the results of that request. All requests and responses are encoded with XML [8] using the DTD found in Section 5. There are two types of requests. One is a general query for a common-name. The other is a request for an object that describes the service and its capabilities. There is only one type of response which is a set of results. Results can contain actual result items, referrals and/or status messages.

3.3 Transport Independence

CNRP is completely encapsulated within its XML definition, and is therefore transport-independent in its specification. However, clients need to have a clearly defined means of bootstrapping a connection with a server.

It is possible to define special-purpose applications that use CNRP but which never need the HTTP bootstrapping method outlined below; those applications MUST define how to find the appropriate server/port/protocol. CNRP servers dedicated to those applications may provide service only on the ports/transport protocols defined by the application.

All other (generic) CNRP clients and servers MUST support the HTTP (Section 7.1) transport on the default CNRP port of 1096.

Note that a particular service may choose to change to a different transport or port via statements within a CNRP service description request, but with initial contacts between a client and a server being over HTTP on port 1096. For a short explanation of how CNRP employs HTTP, see Section 7.1 of this document. If other transports are used, they MUST be handled over a port other than the default CNRP port.

3.4 Character Encoding

To guarantee interoperability, the following provisions apply:

- XML queries and responses MUST be encoded as UTF-8.

  Note: As in any XML document, numeric character references may be used.

- The encoding of characters in the CNRP URI is based on UTF-8; for details, please see [9].

Any interfaces electing to present/accept protocol elements in other representations are responsible for accurate transcoding for use in CNRP protocol elements, per the above provisions.

3.5 Queries

Queries are sent by the client to the server. There are two types of queries.

1. A 'special' initial query that establishes the schema for a particular CNRP database and communicates that to the client. The CNRP client will send this query, and in turn receive an XML document defining the query properties that the database supports. (In CNRP, XML [8] is used to define and express all objects.) This query is called the 'servicequery' in the DTD. In the case where a client does not know anything about the Service, the client MAY assume that it can at least issue the request via HTTP.

2. A 'standard' query, which is the submission of the CNRP search string to the database. The query will conform to the schema that MAY have been previously retrieved from the service.

There will be a set of query properties, listed below, treated as hints by the server. Note: a CNRP database will accept any correctly encoded CNRP query property; the extent to which a query result is responsive to those properties is a service differentiator. The base properties that are always supported are common name, language, geography, category, and range (start and length of the result set). CNRP allows database service providers to create unique data types and expose them to any CNRP client via the CNRP schema XML documents.

3.6 Hints

A hint is an assertion by the user about himself, herself or itself and the context in which he/she/it is operating. There is no data type 'hint'; a hint is expressed within the structure of the query itself and is limited or enabled by the richness of the defined query namespace. In effect, a query and any property within it is a hint.

For example, the "language" property can be given as a hint in a query; this may be used to order search results. If one wants results first in US English followed by European French and finally South American Spanish, the following can be included in the query:

```
<property name="language" type="rfc1766">en-US</property>

<property name="language" type="rfc1766">fr-FR</property>

<property name="language" type="rfc1766">sp-MX</property>
```

Note that the property statements say nothing about whether the language is primary, secondary, etc. In this example, the ordering of the statement controls that--the first statement, being first, means that US English is the primary language. The second statement specifies the second region/language, and so on. *But this is only an example.* The extent to which hints are supported (or not) is a service differentiator.

The fact that a hint exists does not mean that a CNRP database must respond to it. This best-effort approach is similar to relevance ranking in a search engine (high precision, low recall); hints are similar to a search engine's selection criteria. CNRP services will attempt to return the results "closest" to the selection criteria. This is quite different from a SQL database approach where a SQL query returns the entire results set and each result in the set must match all the requirements expressed by the qualifier (the SQL WHERE clause).

4. Object Model

4.1 Properties

In CNRP, objects are property lists. A property is a named attribute. A property also has a well-defined type. Some properties can be part of the query or the results list or both. For simplicity, CNRP is limiting property values to string values.

4.1.1 Core Properties

CNRP introduces a set of core properties. Core properties are the minimal set of properties that all CNRP services MUST support in order to reach CNRP compliance. Hence, the core properties define the level of interoperability between all CNRP services. The core properties are:

1. CommonName: the common name associated with a resource.

2. ID: an opaque string that serves as a unique identifier for a result from a Service (typically a database ID). The ID is not globally unique, nor necessarily persistent (e.g., between queries at a given Service).

3. resourceURI: An 'absoluteURI' as defined in the collected ABNF found in RFC 2396 [3].

4. description: A free text description of the resource.

4.1.2 Abstract and Custom Properties

In addition to core properties, CNRP introduces the notion of abstract properties. The abstract property element provides schema extensibility beyond the core properties. The notion of abstract property is extremely important in CNRP since it enables a wider range of CNRP based services than those based on the core properties.

To create concrete custom properties, a CNRP service must define a property name and a property type. Therefore, there are really two ways to create a custom property. The first way is to create a new property name and define at least one type for it. Another way is to extend an existing property by defining a new type. The "geography" property discussed in the next section is an example of a multi-type property. Note that a type is only applicable to the property it is defined for. If a new property is defined, a new type MUST be defined even though the value set for that type may be identical to an existing type for an existing property. In other words, types are scoped to a given property. Custom properties MUST be registered with IANA. Details about the registration process for new properties can be found in Section 10.

For example, let us assume that a CNRP service specialized on online books would like to introduce the ISBN property of type "number". This property would encapsulate the ISBN number of the book online and would have he following XML representation:

```
<property name="isbn" type="number">92347231</property>
```

4.1.3 Base Properties

Illustrating the use of abstract property to extend the core schema, CNRP also defines a set of custom properties called base properties. In order to keep the requirements extremely simple, these properties are not mandatory to implement to reach CNRP compliance. Although, these properties are not required, it is expected that many services, especially large ones, will implement them. An equally important goal for introducing additional properties is to provide a results filtering mechanism. This is a requirement for large namespaces that contain several million names.

The base properties and their types are defined in Appendix A but listed here for clarity:

- Language:
  The language associated with a resource. The default type of this property is 'RFC1766' and the vocabulary is drawn from the list of languages in RFC 1766 [4]. If RFC 1766 is updated, then the values listed in the updated version are also valid for this type.

- Geography:
  The geographical region or location associated with a resource. Some of the possible types are listed below. See Appendix A for a complete list of types specified by this document.

  * 'freeform': a free form expression for a geographical location (e.g., "palo alto in california").

  * 'ISO3166-1': geographical region expressed using a standard country code as defined by ISO3166-1 (e.g., "US").

  * 'ISO3166-2': value = a geographical region expressed using a standard region and country codes as defined by ISO3166-2 (e.g., "US-CA").

  * 'lat-long': the latitude and longitude of a geographical location.

- Category:
  The category associated with a resource. There are large numbers of possible types for this property. Two possible ones are:

  1. 'freeform': a free form expression for a category (e.g., "movies").

  2. 'NAICS': The North American Industry Code System.

- Range:
  The range is a results set control property. The range property is used to specify the starting point and the length of a results set (e.g., I want 5 records starting at the 10th record). It should only ever have one type but, in the interest of extensibility and consistency, others can be created if there is a need. The default type is 'start-length' which takes the form of two integers separated by a dash. The first integer is the starting number and the second is the number of values to include.

- Dataseturi: An absoluteURI (as defined in [3] that identifies a defined set of Common Names and associated data.

Note: For many properties the default "type" is "freeform". The free form type value is important because it allows very simple user interface where the user can enter a value in a text field. It is up to the service to interpret the value correctly and take advantage of it to increase the relevance of results (using specialized dictionaries for instance).

4.1.4 Common Name String Encoding and Equivalence Rules

CNRP specifies that common name strings should be encoded using UTF-8. CNRP does not specify any string equivalence rules for matching a common name in the query against a common name of a Resource. String equivalence rules are language and service dependent. They are specific to relevance ranking algorithms, hence treated as CNRP services. Consequently, string equivalence rules are not part of the CNRP protocol specification. For example, the query member:

```
<commonname>bmw</commonname>
```

should be read as a selection criterion for a resource with a common name LIKE (similar to) the string "bmw" where the exact definition of the LIKE operator is intuitive, yet specific to the queried CNRP service.

It is also important to note that XML treats whitespace as a special case in many situations. In some cases, it collapses whitespace into a single space. Both client and server Implementors are warned to reference the XML standard for the various ramifications of using whitespace in queries and/or results.

4.2 Objects

4.2.1 Query

The Query object encapsulates all the query components such as CommonName, ID, and any properties. A Query cannot be empty. A Query must contain either one and only one common name, or one and only one ID. A Query can also contain the custom properties defined by a specific CNRP service.

For example, a query for the first 5 resources whose common name is like "bmw" would be expressed as:

```
<query>
        <commonname>bmw</commonname>
        <property name="range" type="start-length">1-5</property>
</query>
```

4.2.1.1 Logical Operations Within a Query

The Query syntax is extremely simple. CNRP does not extensively support Boolean logic operator such as OR, AND or NOT. However, there exist two implicit logical operations that can be expressed through the Query object and its properties. First, a query with multiple property-value pairs implicitly expresses an AND operation on the query terms. For instance, the CNRP query to request all the resources whose common name is like "bmw", AND whose language is "German" can be expressed as:

```
<query>
     <commonname>bmw</commonname>
     <property name="language" type="rfc1766">
        de-DE
     </property>
</query>
```

Note however, that because the server is only trying to best match the Query criteria, there is no guarantee that all or any of the resources in the results match both requirements.

In addition, CNRP allows the client to express a logical OR by specifying multiple values for the same property within the Query. For example, the logical expression:

```
   property = value1 OR property = value2 OR property = valueN
```

Will be expressed as:

```
<property>value1</property>
<property>value2</property>
<property>valueN</property>
```

So if there are different properties expressed, CNRP ANDs them; if there are multiples instances of the same property expressed, CNRP ORs them.

It is important to underline that this form is only applicable to properties (with the exception of the CommonName itself which, even though it is a property, is the entire point of the query). In particular, logical OR operations on the common name are not supported. Note that the ordering or the property-value pairs in the query implies a precedence. As a consequence, CNRP also introduces one special string value: "*". Not surprisingly, "*" means all admissible values for the typed property. For example, the following query requests all the resources whose common name is like BMW and whose language is preferably in German or French or any other language.

```
<query>
     <commonname>bmw</commonname>
     <property name="language" type="rfc1766">de-DE</property>
     <property name="language" type="rfc1766">fr-FR</property>
     <property name="language" type="rfc1766">*</property>
</query>
```

4.2.2 Results

The results object is a container for CNRP results. The type of objects contained in Results can be: ResourceDescriptor, Error, Referral and Schema. Results from a CNRP service are ordered by decreasing relevance. When the results set contains results from multiple CNRP services, the results can no longer be ordered (since relevance ranking is specific to a given service). In that case, however, note that results originating from the same service remain ordered.

4.2.2.1 ResourceDescriptor

The ResourceDescriptor object describes an Internet resource (e.g., a Web page, a person, any object identified by a URI). Therefore, the ResourceDescriptor MUST always include the resourceURI property. The ResourceDescriptor can also contain the commonname, URI, ID (the ID of this entry in the service’s database), description, language, geography, and category of the resource. A ResourceDescriptor can also be augmented using custom properties and can reference a service object to indicate its origin (using the serviceRef element). As with referrals, a resourcedescriptor block can also contain an ID attribute that is used by a status message to refer to a particular resourcedescriptor. Be careful not to confuse this ID with the id tag itself which refers to the database id of the actual database entry.

```
<results>
     <service id="i0">
          <serviceuri>http://cnrp.bar.com/</serviceuri>
     </service>
     <resourcedescriptor id="i1">
          <commonname>bmw</commonname>
          <id>foo.com:234364</id>
          <resourceuri>http://www.bmw.de/</resourceuri>
          <serviceref ref="i0" />
          <description>BMW Motorcycles, International</description>
          <property name="language" type="rfc1766">de-DE</property>
     </resourcedescriptor>
     <referral>
          <serviceref ref="i0" />
     </referral>
</results>
```

4.2.3 Service

The Service object provides an encapsulation of an instance of a CNRP service. A service is uniquely identified through the serviceuri tag which MUST be included in the Service object. A Service object MAY include a a brief textual description of the service. It MAY include datasets, servers and custom properties.

```
<service>
     <serviceuri>http://cnrp.foo.com</serviceuri>
     <description>foo.com is a CNRP service specialized on cocktail
        recipes</description>
</service>
```

The service object MAY also be extended by including existing properties to further describe the service. For instance, a service that focuses on French companies could be expressed as:

```
<service>
     <serviceuri>http://cnrp.foo.com</serviceuri>
     <property name="category" type="freeform">companies</property>
     <property name="geography" type="ISO3166-1">FR</property>
</service>
```

4.2.3.1 Datasets

The dataset object represents a set of CN-to-URI mappings. For example, the database of AOL keywords and their URIs constitute a dataset. The dataset object allows a CNRP implementation to uniquely identify the database(s) of mappings that it resolves. In that respect, the notion of dataset allows a separation between resolution

and data, providing the mechanism for a CNRP service to resolve common-names on behalf of another CNRP service or even multiple services. Conversely, the same dataset can be served by two distinct CNRP services. Since a CNRP service can resolve names within one or more datasets, the service object can contain one or more dataset objects (zero if the dataset is not formally declared).

Within the service object, a dataset is uniquely defined using the dataseturi property. Other properties, such as language and description, can describe the dataset further. Like the service object, the dataset object has an ID attribute associated with it that is unique within a particular XML message. Like the service object's ID attribute, this ID is used by resourcedescriptors and referrals to specify which service and/or dataset they came from or are referring to.

Any service can be said to have a 'default dataset' which is the dataset that considered to have been used if a server simply responds to a client's query that didn't contain a dataset. The 'default dataset' can also be said to be the only dataset that is used by Services that don't support datasets at all. This concept is useful for clients that intend on doing rigorous loop detection by way of keeping a list of visited service/dataset nodes.

This example illustrates how the service object would look as it defines two datasets:

```
<service id="i0">
 <serviceuri>http://acmecorp.com</serviceuri>
 <dataset id="i1">
   <property name="dataseturi">
      urn:oid:1.2.3.4.666.5.4.3.1
   </property>
   <property name="language">en-us</property>
   <property name="language">en-gb</property>
 </dataset>
 <dataset id="i2">
   <property name="dataseturi">
      urn:oid:1.2.3.4.666.10.9.8.7.6
   </property>
   <property name="language">fr</property>
 </dataset>
</service>
```

The dataseturi property can also be used within the query as a hint to the service for the dataset within which the commonname should be resolved:

```
<query>
   <commonname>toys r us</commonname>
   <property name="dataseturi">urn:oid:1.2.3.4.666.5.4.3.1</property>
</query>
```

It is important to note that resolution rules (i.e., string equivalence, relevance ranking, etc.) are likely to be dataset specific. This is true even if the resolution is provided by the same service.

Another use of the dataseturi property is in a referral. In that case, the datasetref tag is used to pinpoint a specific dataset within the service.

```
<referral>
   <serviceref ref="i0" /><datasetref ref="i1" />
</referral>
```

While the concept of datasets is important for services wishing to make their data available via other services, it is important to remember that the declaration and use of datasets is completely optional. Compliance with the CNRP protocol does not require a service object to define or reference any dataset object. The only requirement for compliance is that a client and/or server know the format of the particular XML tags and deal with them syntactically. If it chooses to ignore them, then this is well within its rights.

4.2.3.2 Servers

The service object also encapsulates a list of server objects. The server object is used to describe a CNRP server or set of servers. A server is identified through its serveruri. The URI used to identify a server is not a CNRP URI [9], but instead, is a URI of the scheme used as the CNRP transport mechanism. I.e., for a CNRP server that will communicate via the HTTP protocol to the host foo.com on port 6543, the serveruri would be http://foo.com:6543. If some other information is required in order for the correct transport to be used, then that information can be communicated via other properties. Note that a Service MUST have at least one Server that responds on the default CNRP port in order for a client to get the initial Service object.

A server can serve one or more datasets declared by its service. The served databases are specified using the dataseturi property. As for other objects, a server can be further described using descriptive properties such as geography and description. The following XML completes the service definition from the previous example by defining two CNRP servers. One server is located in the US and the

other is located in France. The US server is specialized and only serves the French dataset.

```
<servers>
   <server>
      <serveruri>cnrp://router.us.widgetco.com:4321</serveruri>
      <property name="geography" type="ISO3166-1">US</property>
   </server>
   <server>
      <serveruri>cnrp://router.fr.acmeco.com:4321</serveruri>
      <property name="geography" type="ISO3166-1">FR</property>
   </server>
</servers>
```

As we will see in a following section, the Service object can contain Schema objects. These Schema objects fully describe the query and response interfaces implemented by a CNRP service. In that regard, the Service object is essential to discoverability. It constitutes the main entry point for a CNRP client to dynamically discover the capabilities of a resolution service. For that purpose, the Service object can be returned as part of the response to any resolution query. Furthermore, the Service object is the dedicated response to the specialized servicequery (see Section 4.2.6).

Another use of Service is for other objects to indicate their CNRP service of origin. System messages, referrals and resourcedescriptors can include a reference to their Service object. For example, imagine a CNRP service that acts as a proxy for multiple CNRP services. For example, it is a requirement that CNRP allows aggregation of results from different sources. Consider one such CNRP service that acts as a proxy for multiple CNRP services. In this mode, the proxy service contacts each CNRP sub-service in parallel or serially. Then, the proxy combines the individual result sets into a unique response returned to the CNRP client. Since the aggregate result set contains resourcedescriptors from different services, the proxy adds a servicereference tag within each individual result to indicate their service of origin. In the event one of the referred services resolves names within multiple datasets, it is possible for these objects to refer to a specific dataset within the service by using the datasetref tag. This example is of a hybrid result set with resourcedescriptors referencing their service and dataset of origin:

```
<?xml version="1.0"?>
<!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
    "http://ietf.org/dtd/cnrp-1.0.dtd">
<cnrp>
     <results>
          <service id="i0">
               <serviceuri>http://acmecorp.com</serviceuri>
               <dataset id="i1">
                  <property name="dataseturi">
                   urn:oid:1.2.3.4.666.5.4.3.1
                  </property>
               </dataset>
               <dataset id="i2">
                  <property name="dataseturi">
                   urn:oid:1.2.3.4.666.10.9.8.7.6
                  </property>
               </dataset>
          </service>
          <service id="i3">
             <serviceuri>http://serverfarm.acmecorp.com</serviceuri>
          </service>
          <service id="i4">
              <serviceuri>http://servers.acmecorp.co.uk</serviceuri>
              <dataset id="i5">
                  <property name="dataseturi">
                    urn:oid:1.2.3.4.666.5.4.3.1
                  </property>
              </dataset>
          </service>
          <resourcedescriptor>
                    <commonname>Fidonet</commonname>
                    <id>1333459455</id>
                    <resourceuri>http://www.fidonet.ca</resourceuri>
                    <serviceref ref="i0" /><datasetref ref="i1" />
                    <description>This is ye olde Canadian
                     Fidonet</description>
          </resourcedescriptor>
          <resourcedescriptor>
                    <commonname>Fidonet</commonname>
                    <id>1333459455</id>
                    <resourceuri>http://host:port/bla</resourceuri>
                    <serviceref ref="i3" />
                    <description>An old Fidonet node</description>
          </resourcedescriptor>
          <referral>
              <serviceref ref="i0" /><datasetref ref="i2" />
          </referral>
     </results>
```

```
</cnrp>
```

4.2.4 Status Messages

4.2.4.1 Status of CNRP, Not the Transport

The status messages defined here are only applicable to operations defined by CNRP itself. If some feature or operation is defined by the transport (security via HTTP, mail failure via SMTP, etc.), then any status messages about that operation MUST be sent in accordance with that transport's reporting mechanism and not via CNRP.

4.2.4.2 Codes and Description

A Status object indicates a message to the client in the results set. The object encapsulates two values: a status code and a description. The description can contain a textual description of the status being communicated. In many cases, additional diagnostic information can also be included. No attempt is made to standardize the description of a given status code since the only programmatic element that matters is the actual code.

A status message can also specify which other CNRP element it refers to by including a reference to the ID of the element in question. For example, if a Service block has an ID of "i2" and a status message refers to that block, then it can put that ID in its ref attribute.

```
<status code="x.y.z" ref="i2">
     The CNRP foo.com database is temporarily unreachable
</status>
```

4.2.4.3 Status Codes

The organization of status codes is taken from RFC 1893 [10] which structures its codes in the form of x.yyy.zzz. Taken from RFC 1893 is the ABNF for the codes:

```
status-code = class "." subject "." detail
class = "2"/"3"/"4"/"5"
subject = 1*3digit
detail = 1*3digit
```

The top level codes denote levels of severity of the status:

- 1.X.X Informational
  - The information conveyed by the code has no bearing or indication of the success or failure of any request. It is strictly for informational purposes only.
- 2.X.X Success
  - The request was processed and results were returned. In most cases, this status class won't be sent since actual results themselves denote success. In other cases, results were returned but some information needs to be returned to the client.
- 3.X.X Partial Success
  - The request was processed and results were returned. In this case though, some values sent with the request were either invalid or ignored but in a way that the server still considers the response to be a successful one and not indicative of any true error condition.
- 4.X.X Transient Failure
  - The request was valid as sent, but some temporary event prevents the successful completion of the request and/or sending of the results. Sending in the future may be possible.
- 5.X.X Permanent Failure
  - A permanent failure is one which is not likely to be resolved by re-sending the request in its current form. Some change to the request or the destination must be made for successful request.

The second level codes denote the subject of the status messages. This value applies to each of the five classifications. The subject sub-code, if recognized, must be reported even if the additional detail provided by the detail sub-code is not recognized. The enumerated values for the subject sub-code are:

- X.0.X Other or Undefined Status
  - No specific information is available about what subject class this message belongs to.
- X.1.X Query Related
  - Any status related to some specific way in which the query was encoded or its values with the exception of properties.
- X.2.X Service Related
  - Any status related to the service in which this server is cooperating in providing.

Appendix B contains a list of all predefined status codes

4.2.5 Referral

A Referral object in the results set is a place holder for un-fetched results from a different service and possibly dataset. Referrals typically occur when a CNRP server knows of another service capable of providing relevant results for the query and wants to notify the client about this possibility. The client can decide whether it wants to follow the referral and resolve the extra results by contacting the referred-to service using the information contained within the Referral object (a Service object and possible properties). The Referral is a simple mechanism to enable hierarchical resolution as well as to join multiple resolution services together.

```
<results>
     <service id="i0">
          <serviceuri>http://cnrp.bar.com/</serviceuri>
          <dataset id="i1">
             <property name="dataseturi">
               urn:oid:1.3.6.1.4.1.782.1
             </property>
          </dataset>
          <dataset id="i2">
             <property name="dataseturi">
               urn:oid:1.3.6.1.4.1.782.2
             </property>
          </dataset>
     </service>
     <resourcedescriptor>
          <commonname>bmw</commonname>
          <id>foo.com:234364</id>
          <resourceuri>http://www.bmw.de/</resourceuri>
          <serviceref ref="i0" /><datasetref ref="i1" />
          <description>BMW Motorcycles, International</description>
          <property name="language" type="iso646">de-DE</property>
     </resourcedescriptor>
     <referral>
          <serviceref ref="i0" /><datasetref ref="i2" />
     </referral>
</results>
```

Like other CNRP objects, a referral can be further described using custom properties. Like a resourcedescriptor, a referral can have an ID attribute that is used by a status message to talk about a particular referral block.

4.2.5.1 Loop Detection and Dataset Handling in Servers

Referrals in CNRP can be handled in three ways:

- application specific,
- as hints only,
- rigorous loop detection.

In the first two cases, the behavior of the client, when it receives a referral, is not defined in this memo. The client can chase the referral in such a way as to treat it as a hint only. In this case, datasets may or may not be handled. Loop detection can be nothing more than, "Have I talked to this hostname before?" or "Stop after the 3rd referral". These two cases are most likely to apply to

simple or constrained implementations where the clients and servers have some a priori knowledge of their capabilities. Without such knowledge there is too much ambiguity vis-a-vis services and datasets for clients to do reliable loop detection.

The last case is where the client expects to talk to multiple servers that may know nothing about each other. This case expresses the basic semantics of what a server should tell a client if it understands datasets or referrals. Since a referral specifies the exact dataset to which it is referring, a node in the list of visited nodes is made up of a serviceuri and a dataseturi. Both of these values need to be considered during loop detection. In the case where a service does not support datasets, the visited node is made up of the service and the 'default dataset'.

The major thing to remember when doing loop detection across servers is that some servers may not understand datasets at all, while others specifically rely on them. To help determine how loop detection nodes should be marked, three specific status messages have been defined:

The 3.1.3 (Datasets not supported) status message is used to denote that the server does not support datasets at all. It is sent in response to a query containing datasets. The client should consider that the server ignored the datasets and the client should consider this node to have been visited for all possible datasets (including the 'default' dataset).

The 3.1.4 (First dataset only supported) status message is used by a server to indicate the situation where a client has included several datasetуris in its query and the server can only support one at a time. In this case, the server is explicitly stating that it used the first dataseturi only. The client should consider that only the first dataseturi specified was processed correctly. The client should consider that the remaining datasets in the query were ignored completely. They would need to be sent individually as referrals if the client really cares about those results. Only the first serviceuri/dataseturi pair should be marked as visited.

The 3.1.5 (This dataset not supported) status message is used to indicate that a specific dataseturi sent in a query by a client is not supported by the server. This serviceuri/dataseturi pair should be considered as visited by the client. If this message is sent in reply to a query specifying multiple datasets, the client should behave the same as if it received the 3.1.3 message from above. It should be considered bad form for a server to send this status message back in response to a query with multiple datasets because it is ambiguous.

While there is no exact algorithm for loop detection that clients are encouraged to support, these status messages can be used by the server to be clear about what Services and Datasets it considers to have been queried. It is up to the client to decide what to do with these messages and how closely it attempts to do loop detection.

4.2.6 Discoverability: ServiceQuery and Schema

A subclass of Query, the ServiceQuery object supports the dynamic discovery of a specific CNRP service's characteristics. Note that CNRP compliance does not require that a service fully implements discoverability. In particular, returning the Service object with its serviceuri constitutes a minimal yet sufficient compliant implementation. Nevertheless, we expect that advanced CNRP services will choose to return a full description of their supported interfaces.

The complete response to a servicequery returns the Service object described in section 5.3.2 with the following schema information:

1. The base and custom properties used by the CNRP service (Property schema),

2. The properties used to describe the Service object (Service schema),

3. The properties that belong to the query interface (Query schema),

4. The properties that belong to a resource within the results (Resource schema).

These leads to the following new object definitions:

- propertyschema -- A property schema describes all the custom properties that are part of the service.

- propertydeclaration -- A property declaration describes a base or custom property used by the CNRP service. A property declaration has a name and a type (the name and the type of the property that it refers to). Note that as part of the property schema, one MUST declare both existing and newly defined properties.

- propertyreference -- A property reference is a reference to a property declaration so that a given schema (a service, query or resource schema) can declare the property within its interface. Note that a property reference specify whether the use of the property is required or optional only.

- serviceschema -- The service schema defines the properties used to describe the service.

- queryschema -- A query schema describes the structure of a query handled by the CNRP service. The properties referred within the query schema are part of the query interface of the resolution service.

- resourcedescriptorschema -- A ResourceDescriptor schema describes the resource returned as a result by the CNRP service.

For example, a CNRP query to discover a service's capabilities will be in the form:

```
<cnrp> <servicequery/> </cnrp>
```

And for a CNRP service for cocktail recipes in French, the corresponding response would be:

```
<service>
     <serviceuri>http://cnrp.recipe.com</serviceuri>
     <propertyschema>
        <propertydeclaration id="i1">
              <propertyname>language</propertyname>
              <propertytype>rfc1766</propertytype>
        </propertydeclaration>
        <propertydeclaration id="i2">
              <propertyname>cocktailrecipe</propertyname>
              <propertytype>freeform</propertytype>
        </propertydeclaration>
     </propertyschema>
     <queryschema>
          <propertyreference required="yes" ref="i1"/>
     </queryschema>
     <resourcedescriptorschema>
          <propertyreference required="yes" ref="i1"/>
          <propertyreference required="yes" ref="i2"/>
     </resourcedescriptorschema>
</service>
```

This response stipulates that the service accepts the property language as part of the query interface and returns resourcedescriptors that contain both the language and cocktailRecipe properties.

5. XML DTD for CNRP

```
<!-- The document tag -->
<!ELEMENT cnrp (query|results|servicequery)>

<!-- Used to request a Service object -->
<!ELEMENT servicequery EMPTY>

<!-- A query can either request a schema, a specific record by -->
<!-- id, or a common-name with a set of properties (or         -->
<!-- assertions) about the entity doing the query.             -->
<!ELEMENT query (id|(commonname,property*))>
<!ELEMENT id (#PCDATA)>

<!ELEMENT commonname (#PCDATA)>
<!-- NOTE: CNRP defines several well known properties          -->
<!-- and types. See Appendix A for details.                    -->
<!ELEMENT property (#PCDATA)>
<!-- The name of the property -->
<!ATTLIST property name CDATA #REQUIRED>
<!-- The type of the property -->
<!ATTLIST property type CDATA "freeform">

<!ELEMENT results (status? |
                   ( service+,
                        ( status  | resourcedescriptor | referral )*
                   )*
                  )>

<!ELEMENT resourcedescriptor (commonname,id,resourceuri,
    serviceref, datasetref?,
    description,
    property*)>
<!ATTLIST resourcedescriptor id ID #IMPLIED>

<!-- The entire point of all this... -->
<!ELEMENT resourceuri (#PCDATA)>
<!ELEMENT description (#PCDATA)>

<!ELEMENT referral (serviceref, datasetref?)>
<!ATTLIST referral id ID #IMPLIED>

<!ELEMENT status (#PCDATA)>
<!ATTLIST status code CDATA #REQUIRED>
<!ATTLIST status ref IDREF #IMPLIED>

<!-- serviceRef is used to point to one of a set of provided   -->
<!-- service objects. This is so that a resource can point to  -->
```

```
   <!-- which service it came from. We could include the entire   -->
   <!-- service object but then we would be repeating large       -->
   <!-- amounts of information.                                   -->

   <!ELEMENT serviceref EMPTY>
   <!ATTLIST serviceref ref IDREF #IMPLIED>

   <!ELEMENT service (serviceuri, dataset*,
      servers?,
      description?,
      property*,propertyschema?,queryschema?,resourcedescriptorschema?,
      serviceschema?)>
   <!-- The time to live of the schema in seconds since it was   -->
   <!-- retrieved -->
   <!ATTLIST service ttl CDATA "0">
   <!ATTLIST service id ID #IMPLIED>
   <!ELEMENT serviceuri (#PCDATA)>
   <!ELEMENT servers (server+)>
   <!ELEMENT server (serveruri, property*)>
   <!ELEMENT serveruri (#PCDATA)>

   <!ELEMENT dataset (property*)>
   <!ATTLIST dataset id ID #IMPLIED>

   <!ELEMENT datasetref EMPTY>
   <!ATTLIST datasetref ref IDREF #IMPLIED>

   <!ELEMENT propertyschema (propertydeclaration*)>
   <!ELEMENT propertydeclaration (propertyname, propertytype*)>
   <!ATTLIST propertydeclaration id ID #IMPLIED>

   <!ELEMENT propertyname (#PCDATA)>
   <!ELEMENT propertytype (#PCDATA)>
   <!-- This specifies if the type is meant to be the default -->
   <!-- type. This is usually reserved for "freeform".        -->
   <!ATTLIST propertytype default (no|yes) "no">

   <!-- The properties you can use in a query -->
   <!ELEMENT queryschema (propertyreference*)>

   <!-- The properties you can expect to see in an Resource -->
   <!ELEMENT resourcedescriptorschema (propertyreference*)>

   <!-- The properties you can expect to find in a Service  -->
   <!-- definition -->
   <!ELEMENT serviceschema (propertyreference*)>

   <!ELEMENT propertyreference EMPTY>
```

```
   <!-- This specifies if a property is required as part of -->
   <!-- the query. -->
   <!ATTLIST propertyreference ref IDREF #REQUIRED>
   <!ATTLIST propertyreference required (no|yes) "no">
```

6. Examples

6.1 Service Description Request

This is what the client sends when it is requesting a servers schema.

```
<?xml version="1.0"?>
<!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
 "http://ietf.org/dtd/cnrp-1.0.dtd">
<cnrp>
     <servicequery />
</cnrp>
```

This is the result. Notice how the Service tag is used to allow the service to describe itself in its own terms.

```
<?xml version="1.0"?>
<!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
 "http://ietf.org/dtd/cnrp-1.0.dtd">
<cnrp>
 <results>
  <service ttl="43200">
    <serviceuri>urn:foo:bar</serviceuri>
    <servers>
      <server>
          <serveruri>http://host1.acmecorp.com:4321/foo?</serveruri>
      </server>
      <server>
          <serveruri>smtp://host2.acmecorp.com:4321/foo?</serveruri>
      </server>
    </servers>
    <description>This is the Acme CNRP Service</description>
    <!-- This property means that Acme specializes in
         tradename services -->
    <property name="category" type="naics">544554</property>
    <property name="BannerAdServer" type="uri">
              http://adserver.acmecorp.com/
    </property>
    <propertyschema>
      <propertydeclaration id="i1">
        <propertyname>workgroupID</propertyname>
        <propertytype default="yes">freeform</propertytype>
        <propertytype default="no">domainname</propertytype>
```

```
        </propertydeclaration>
        <propertydeclaration id="i2">
          <propertyname>BannerAdServer</propertyname>
          <propertytype default="yes">URI</propertytype>
        </propertydeclaration>
      </propertyschema>
      <queryschema>
          <propertyreference ref="i1" required="yes" />
      </queryschema>
      <resourcedescriptorschema>
          <propertyreference ref="i1" required="yes" />
      </resourcedescriptorschema>
      <serviceschema>
          <propertyreference ref="i2" required="yes" />
      </serviceschema>
    </service>
   </results>
  </cnrp>
```

6.2 Sending A Query and Getting A Response

This is the query that is sent from the client to the server:

```
<?xml version="1.0"?>
<!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
 "http://ietf.org/dtd/cnrp-1.0.dtd">
<cnrp>
 <query>
    <commonname>Fido</commonname>
    <property name="geography" type="iso3166-2">
       CA-QC</property>
    <property name="geography" type="iso3166-1">CA</property>
    <property name="language" type="rfc1766">fr-CA</property>
 </query>
</cnrp>
```

This is the result set. It is sent back in response to the query. This result set includes a referral and a non-fatal error.

```
<?xml version="1.0"?>
<!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
 "http://ietf.org/dtd/cnrp-1.0.dtd">
<cnrp>
  <results>
    <service id="i0">
      <serviceuri>http://acmecorp.com</serviceuri>
    </service>
    <service id="i1">
```

```
        <serviceuri>http://serverfarm.acmecorp.com</serviceuri>
      </service>
      <service id="i2">
        <serviceuri>http://servers.acmecorp.co.uk</serviceuri>
      </service>
      <resourcedescriptor>
        <commonname>Fidonet</commonname>
        <id>1333459455</id>
        <resourceuri>http://www.fidonet.ca</resourceuri>
        <serviceref ref="i0" />
        <description>This is ye olde Canadian Fidonet</description>
      </resourcedescriptor>
      <resourcedescriptor>
        <commonname>Fidonet</commonname>
        <id>1333459455</id>
        <resourceuri>http://host:port/bla</resourceuri>
        <serviceref ref="i1" />
        <description>An old Fidonet node</description>
      </resourcedescriptor>
      <referral><serviceref ref="i2" /></referral>
      <status code="3.1.1">
          The language property 'fr-CA' was ignored
      </status>
    </results>
  </cnrp>
```

7. Transport

Two CNRP transport protocols are specified. HTTP is used due to its popularity and ease of integration with other web applications. SMTP is also used as a way to illustrate a protocol that has a much different range of latency than most protocols.

In the cases where transports use MIME Media Types (HTTP and SMTP being examples of such), the CNRP payload MUST use the 'application/cnrp+xml' media type. See Section 8 for the registration template for this media type. One important note about this media type is that, since CNRP always uses UTF-8, there is no charset attribute.

7.1 HTTP Transport

The HTTP transport is fairly simple. The client connects to an HTTP based CNRP server and issues a request using the POST method to the "/" path with the Content-type and Accept header set to "application/cnrp+xml". The content of the POST body is the CNRP XML document that is being sent. All HTTP 1.1 features are allowed during the request.

The results are sent back to the client with a Content-Type of "application/cnrp+xml". The body of the result is the CNRP XML document being sent to the client.

7.2 SMTP Transport

The SMTP transport is very similar to the HTTP transport. Since there is no method to specify, the CNRP XML document is simply sent to a particular SMTP endpoint with its Content-Type set to "application/cnrp+xml". The server responds by sending a response to the originator of the request with the results in the body and the Content-Type set to "application/cnrp+xml". The Service MUST specify at least one SMTP target (email address) to contact.

8. Registration: application/cnrp+xml

This is the registration template for 'application/cnrp+xml' per [6].

MIME media type name: application

MIME subtype name: cnrp+xml

Required parameters: none

Optional parameters: none

Encoding considerations: This media type consists of 8bit text which may necessitate the use of an appropriate content transfer encoding on some transports. Since these considerations are the same as XML in general, RFC3023's [6] discussion of XML and MIME is applicable.

Security considerations: none specific to this media type. See Section 9 for general CNRP considerations.

Interoperability considerations: n/a

Published specification: This media type is a proper subset of the the XML 1.0 specification [8] except for the limitations placed on tags and encodings by this document.

Applications which use this media type: any CNRP client/server wishing to send or receive CNRP requests or responses

Additional Information: none

Contact for further information: c.f., the "Author's Address" section of this memo

Intended usage: limited use

Author/Change controller: the IESG

9. Security Considerations

Three security threats exist for CNRP or applications that depend on it: Man in the Middle attacks, malicious agents posing as a service by spoofing a Service object, and denial of service attacks caused by adding a new level of indirection for resolution of a resource.

The proposed solution for man in the middle attacks is to utilize transport level authentication and encryption, where available. In the case where the transport can't provide the level of required authentication, individual entries or the entire response can be signed/encrypted using XML signature methods being developed by the XMLDSIG Working Group.

In the case of where a service attempts to pose as another by spoofing the serviceuri in the Service object, the Service object should be signed. A client can then verify the Service object's veracity by verifying the signature. How the client obtains that authoritative public key is out of scope since it depends on the service discovery problem.

While this document cannot propose a solution for Denial Of Service (DOS) attacks, it can illustrate that, like many other cases, any time a new level of indirection is created, an opportunity for a DOS attack is created. Service providers are encouraged to be aware of this and to act accordingly to mitigate the effects of a DOS attack.

10. IANA Considerations

The major consideration for the IANA is that the IANA will be registering well known properties, property types and status messages. It will not register values. Since this document does not discuss CNRP service discovery, the IANA will not be registering the existence of servers or Server objects.

There are three types of entities the IANA can register: properties, property types, and status messages. If a property or type is not registered with the IANA, then they must start with "x-". Status messages can be created for local consumption and not registered. There is no requirement that new status messages are mandatory to implement unless this document is updated. Status message registrations are more for informational purposes.

The required information for the registration of a new property is the property's name, its default type, and a general description. A new type requires the type's name, what properties it is valid for, and a description. A new status message requires the X.Y.ZZZ code and a brief description of the state being communicated.

All properties, types and status messages are registered on a First Come First Served basis with no review by the IANA or any group of experts. The consensus opinion of the CNRP Working Group is that review of property registrations should occur once there is operational experience with the protocol and an actual need for the review. If, at some future date, this policy needs to change, this document will be updated.

The property and type registration templates found in Appendix A should be registered by the IANA at publication time of this document.

The IANA is also directed to register the Media Type specified in Section 8.

References

[1] United States, "North American Industry Classification System", January 1997, <http://www.census.gov/epcd/www/naics.html>.

[2] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[3] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[4] Alvestrand, H., "Tags for the Identification of Languages", RFC 1766, March 1995.

[5] Moats, R., "URN Syntax", RFC 2141, May 1997.

[6] Murata, M., St. Laurent, S. and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[7] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[8] Bray, T., Paoli, J. and C. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0", February 1998.

[9] Mealling, M., "The 'go' URI Scheme for the Common Name Resolution Protocol", RFC 3368, August 2002.

[10] Vaudreuil, G., "Enhanced Mail System Status Codes", RFC 1893, January 1996.

[11] "Country and Region Codes", ISO 3166, January 1996.

Appendix A. Well Known Property and Type Registration Templates

A.1 Properties

Property Name: geography
Default Type: iso3166-1
Description: A geographic location

Property Name: language
Default Type: rfc1766
Description: A language specification

Property Name: category
Default Type: freeform
Description: A node in some system of semantic relationships that is considered relevant to the common-name.

Property Name: range
Default Type: range
Description: A range given in the format "x,y" where x is the starting point and y is the length. This property is used by the client to tell the server that is is requesting a subrange of the results.

Property Name: dataseturi
Default Type: uri
Description: A URI used to disambiguate between two Datasets offered by the same Service.

A.2 Types

Type: freeform
Property: category
Description: The value is to be interpreted by the server the best way it knows how. This value has no defined structure.

Type: freeform
Property: geography
Description: The value is to be interpreted by the server the best way it knows how. This value has no defined structure.

Type: freeform
Property: language
Description: The value is to be interpreted by the server the best way it knows how. This value has no defined structure.

Type: iso3166-2
Property: geography
Description: The combination of country and sub-region codes found in ISO 3166-2 [11].

Type: iso3166-1
Property: Geography
Description: Country Codes found in ISO 3166-1 [11].

Type: postalcode
Property: Geography
Description: A postal code that is valid for some region. A good example is the Zip code system used in the US.

Type: lat-long
Property: Geography
Description:

Values for latitude and longitude shall be expressed as decimal fractions of degrees. Whole degrees of latitude shall be represented by a two-digit decimal number ranging from 0 through 90. Whole degrees of longitude shall be represented by a decimal number ranging from 0 through 180. When a decimal fraction of a degree is specified, it shall be separated from the whole number of degrees by a decimal point. Decimal fractions of a degree may be expressed to the precision desired.

Latitudes north of the equator shall be specified by a plus sign (+), or by the absence of a minus sign (-), preceding the designating degrees. Latitudes south of the Equator shall be designated by a minus sign (-) preceding the two digits designating degrees. A point on the Equator shall be assigned to the Northern Hemisphere.

Longitudes east of the prime meridian shall be specified by a plus sign (+), or by the Longitudes west of the meridian shall be designated by minus sign (-) preceding the digits designating degrees. A point on the prime meridian shall be assigned to the

Eastern Hemisphere. A point on the 180th meridian shall be assigned to the Western Hemisphere. One exception to this last convention is permitted. For the special condition of describing a band of latitude around the earth, the East Bounding Coordinate data element shall be assigned the value +180 (180) degrees.

Any spatial address with a latitude of +90 (90) or -90 degrees will specify the position at the North or South Pole, respectively. The component for longitude may have any legal value.

With the exception of the special condition described above, this form is specified in Department of Commerce, 1986, Representation of geographic point locations for information interchange (Federal Information Processing Standard 70-1): Washington, Department of Commerce, National Institute of Standards and Technology.

```
DEGREES   = *PLUSMINUS DIGITS '.' DIGITS
PLUSMINUS = + | -
DIGITS    = DIGIT *DIGIT
DIGIT     = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Type: rfc1766
Property: Language
Description: language codes as defined by RFC 1766 [4]

Type: naics
Property: Category
Description: North American Industry Code System [1]

Type: uri
Property: dataseturi
Description: A URI adhering to the 'absoluteURI' production of the Collected ABNF found in [3]

Appendix B. Status Codes

B.1 Level 1 (Informative) Codes

1.0.0 -- Undefined Information
This code is used for any non-categorizable and informative message. If, for example, the server wanted to tell the client that the systems administrator's cat has blue hair, then this code would be the appropriate place for this information.

1.1.0 -- Query related information
This code is used for any informative information concerning the query that client sent. For example, "The query you sent was rather interesting!".

1.2.0 -- An informative message pertaining to the Service
This message concerns the Service in the general sense.

B.2 Level 2 (Success) Codes

2.0.0 -- Something undefined succeeded
There was success but the situation that this message concerns is undefined.

2.1.0 -- Query succeeded
The query succeeded. This message MUST be returned when there were no results that matched the query. I.e., the query was successfully handled and the correct set of results contained no resources or referrals. The lack of results is not an error but a successful statement about the common-name.

Note: The apparent lack of 2.X.X level codes is caused by success usually being indicated not by a status message but by the server returning only the objects that the client requested.

B.3 Level 3 (Partial Success) Codes

3.0.0 -- Something undefined was only partially successful
Some request by the client was only partially successful. The exact situation or cause of that partial failure is not defined.

3.1.0 -- The query was only partially successful.

3.1.1 -- The query contained invalid or unsupported properties
The query contained invalid or unsupported property names, types or values. The invalid properties were ignored and the query processed.

3.1.2 -- The XML was well formed but invalid
The XML sent by the client was well formed but invalid. The server was smart enough to figure out what the client was talking about and return some results.

3.1.3 Server does not support datasets

This status should be generated by servers that do not handle datasets. A server can send this status message at any time, but it especially useful for when a server receives a query from a client that contains a dataseturi. In this case and if the client is doing rigorous loop detection, the client should consider this entire service to have been visited.

3.1.4 The first dataset in the list of datasets you gave in the query was the only one used.

This status message is used by a server to indicate the situation where a client has included several dataseturis in its query and the server can only support one at a time. In this case the server is explicitly stating that it used the first dataseturi only. The client should consider that only the first dataseturi specified was processed correctly. The client should consider that the remaining datasets in the query were ignored completely.

They would need to be sent individually as referrals if the client really cares about those results. Only the first serviceuri/dataseturi pair should be marked as visited if loop detection is being handled.

3.1.5 This dataset not supported.

This message is used to indicate that a specific dataseturi sent in a query by a client is not supported by the server. This serviceuri/dataseturi pair should be considered as visited by the client. If this message is sent in reply to a query specifying multiple datasets, the client should behave the same as if it received the 3.1.3 message from above. It should be considered bad form for a server to send this status message back in response to a query with multiple datasets because it is ambiguous.

3.2.0 -- The server caused a partially successful event

Due to some internal server error, the results returned were incomplete.

3.2.1 -- Some referral server was unavailable

This status message is used to denote that one or more of the referral services that are normally queried was unavailable. Results were generated, but they may not be representative of a complete answer.

B.4 Level 4 (Transient Failure) Codes

4.0.0 -- Something undefined caused a persistent transient failure.

4.1.0 -- There was an error in the query that made it unable to be interpreted.

4.2.0 -- The query was to complex
The query as specified was too complex for this Service to handle.

4.2.1 -- The Service was too busy
Due to resource constraints, the entire service is too busy to handle requests. This means that any of the Servers cooperating in providing this Service would have also returned this same message.

4.2.2 -- The Server is in maintenance
This server is now in maintenance mode. Try another server from this service or try again at a later time.

4.2.3 -- The Server had an internal error
There was an internal error that caused the server to fail completely.

B.5 Level 5 (Permanent Failures) Codes.

5.0.0 -- Something undefined caused a permanent failure.

5.1.0 -- The query permanently failed.

5.2.0 -- The service had a permanent failure.

5.2.1 -- This Service is no longer available.
This Service has decided to no longer make itself available.

5.2.2 -- The Server had a permanent failure.
This server has permanently failed. Try another server from this service.

Authors' Addresses

Nico Popp
VeriSign, Inc.
487 East Middlefield Road
Mountain View, CA 94043

Phone: (650) 426-3291
EMail: npopp@verisign.com

Michael Mealling
VeriSign, Inc.
21345 Ridgetop Circle
Sterling, VA 20166
US

EMail: michael@verisignlabs.com

Marshall Moseley
Network, Inc.
702 Russell Avenue
Gaithersburg, MD 20877-2606
US

Phone: (240) 631-1100
EMail: marshall@network.com

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.