Network Working Group
Internet-Draft
Expires: December 24, 1999

M. Mealling
Network Solutions, Inc.
R. Daniel
DATAFUSION, Inc.
June 25, 1999

# Resolution of Uniform Resource Identifiers using the Domain Name System
## draft-ietf-urn-dns-rds-01

Status of this Memo



Abstract

The architectural principles laid out in RFC2276[9] defines the concept of a "resolver discovery service". This document describes an immediately-deployable RDS. It is implemented by a new DNS Resource Record, NAPTR (Naming Authority PoinTeR)[10], that provides a method for encoding incrementally discovered rules within DNS. By using these incrementally discovered rules to re-map parts of a URI, we can change the host that is contacted to resolve a URI. This will allow a more graceful handling of URLs over long time periods, and forms the foundation for a new proposal for Uniform Resource Names.

In addition to locating resolvers, the NAPTR provides for other naming systems to be grandfathered into the URN world, provides independence between the name assignment system and the resolution protocol system, and allows multiple services (Identifier to Location, Identifier to Description, Identifier to Resource, ...) to

be offered. In conjunction with the SRV RR, the NAPTR record allows those services to be replicated for the purposes of fault tolerance and load balancing.



Table of Contents

## 1. Introduction

Uniform Resource Locators have been a significant advance in retrieving Internet-accessible resources. However, their brittle nature over time has been recognized for several years. The Uniform Resource Identifier working group proposed the development of Uniform Resource Names to serve as persistent, location-independent identifiers for Internet resources in order to overcome most of the problems with URLs. RFC-1737[1] sets forth requirements on URNs.

During the lifetime of the URI-WG, a number of URN proposals were generated. The developers of several of those proposals met in a series of meetings, resulting in a compromise known as the Knoxville framework. The major principle behind the Knoxville framework is that the resolution system must be separate from the way names are assigned. This is in marked contrast to most URLs, which identify the host to contact and the protocol to use. Readers are referred to [2] for background on the Knoxville framework and for additional information on the context and purpose of this proposal.

Separating the way names are resolved from the way they are constructed provides several benefits. It allows multiple naming approaches and resolution approaches to compete, as it allows different protocols and resolvers to be used. There is just one problem with such a separation - how do we resolve a name when it can't give us directions to its resolver?

For the short term, DNS is the obvious candidate for the resolution framework, since it is widely deployed and understood. However, it is not appropriate to use DNS to maintain information on a per-resource basis. First of all, DNS was never intended to handle that many records. Second, the limited record size is inappropriate for catalog information. Third, domain names are not appropriate as URNs.

Therefore our approach is to use DNS to locate "resolvers" that can provide information on individual resources, potentially including the resource itself. To accomplish this, we "rewrite" the URI into a domain name following the rules found in NAPTR records. Rewrite rules provide considerable power, which is important when trying to meet the goals listed above. However, collections of rules can become difficult to understand. To lessen this problem, the NAPTR rules are *always* applied to the original URI, *never* to the output of previous rules.

## 2. 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.

## 3. Overview of the NAPTR Record

The NAPTR record is defined in RFCXXXX[10]. To summarize, the key fields in the NAPTR RR are Order, Preference, Service, Flags, Regexp, and Replacement:

- The order field specifies the order in which records MUST be processed when multiple NAPTR records are returned in response to a single query. A naming authority may have delegated a portion of its namespace to another agency. Evaluating the NAPTR records in the correct order is necessary for delegation to work properly.

- The preference field specifies the order in which records SHOULD be processed when multiple NAPTR records have the same value of "order". This field lets a service provider specify the order in which resolvers are contacted, so that more capable machines are contacted in preference to less capable ones.

- The service field specifies the resolution protocol and resolution service(s) that will be available if the rewrite specified by the regexp or replacement fields is applied. Resolution protocols are the protocols used to talk with a resolver. They will be specified in other documents, such as [5]. Resolution services are operations such as I2R (URI to Resource), I2L (URI to URL), I2C (URI to URC), etc. These are specified in the URI Resolution Services document[6], and their behavior in a particular resolution protocol will be given in the specification for that protocol (see [5] for a concrete example).

- The flags field contains modifiers that affect what happens in the next DNS lookup, typically for optimizing the process. Flags may also affect the interpretation of the other fields in the record, therefore, clients MUST skip NAPTR records which contain an unknown flag value.

- The regexp field is one of two fields used for the rewrite rules, and is the core concept of the NAPTR record. The regexp field is a String containing a sed-like substitution expression. (The actual grammar for the substitution expressions is given later in this draft). The substitution expression is applied to the original URN to determine the next domain name to be queried. The regexp field should be used when the domain name to be generated is conditional on information in the URI. If the next domain name is always known, which is anticipated to be a common occurrence, the replacement field should be used instead.

- The replacement field is the other field that may be used for the rewrite rule. It is an optimization of the rewrite process for

the case where the next domain name is fixed instead of being conditional on the content of the URI. The replacement field is a domain name (subject to compression if a DNS sender knows that a given recipient is able to decompress names in this RR type's RDATA field). If the rewrite is more complex than a simple substitution of a domain name, the replacement field should be set to . and the regexp field used.

Note that the client applies all the substitutions and performs all lookups, they are not performed in the DNS servers. Note also that it is the belief of the developers of this document that regexps should rarely be used. The replacement field seems adequate for the vast majority of situations. Regexps are only necessary when portions of a namespace are to be delegated to different resolvers. Finally, note that the regexp and replacement fields are, at present, mutually exclusive. However, developers of client software should be aware that a new flag might be defined which requires values in both fields.

## 4. The Distinction between URNs and URLs

From the point of view of this system, there is no theoretical difference between resolving URIs in the general case and URNs in the specific case. Operationally however, there is a difference that stems from the unknown case of URI resolution not becoming widespread. If URN resolution is collapsed into generic URI resolution, URNs may suffer by the lack of adoption of URI resolution. The technically correct solution however should discourage such a case.

The solution is to allow for shortcutting for URN resolution. In the following specification generic URI resolution starts by inserting rules for known URI shemes into the 'uri.net' registry. For URN resolution one of the rules would be for the 'urn' URI scheme. This rule would simply delegate to the 'urn.net' zone for additional NAPTRS based on the URN namespace.

Since this rule is the basis for the entire URN RDS, it can be shortcutted by simply starting URN resolution at the 'urn.net' registry. This the distinction between the 'uri.net' and 'urn.net' well known keys seen below.

## 5. The DNS RDS Algorithm

Since the general RDS framework was the basis for the original NAPTR algorithm, the two match very well. The only pieces missing from the general NAPTR specification are the original key, protocols and services.

### 5.1 The First Known Key

In the generic URI case, the first known key is created by taking the URI scheme and appending 'uri.net' to the end. In the specific, shortcutted URN case, the first known key is created by taking the Namespace Identifier and appending 'urn.net' to the end.

#### 5.1.1 URI Example

http://www.foo.com/ would have a first known key of 'http.uri.net'.

#### 5.1.2 URN Example

urn:foo:12345 would have a first known key of 'foo.urn.net'.

### 5.2 Services

The services that make sense for URI resolution are generic for both URI and URN resolution since the input value types itself based on the URI scheme. Some valid services are defined in RFCXXXX (draft-ietf-urn-resolution-services-07.txt).

Examples of some of these services are:

I2L: given a URI return one URL that identifies a location where the original URI can be found

I2Ls: given a URI return one or more URLs that identify multiple locations where the original URI can be found

I2R: given a URI return one instance of the resource identified by that URI.

I2Rs: given a URI return one or more instances of the resources identified by that URI.

I2C: given a URI return one instance of a description of that resource.

I2N: given a URI return one URN that names the resource (Caution: equality with respect to URNs is non-trivial. See [9] for

examples of why.)

## 5.3 Protocols

The protocols used in the Services field are currently limited to THTTP[5]. Simply specifying any protocol in the services field is insufficient since there are additional semantics surrounding URI resolution that are not specified within the protocols.

For example, if Z39.50 were to be specified as a valid protocol it would have to define how it would encode requests for specific services, how the URI is encoded, and what information is returned.

Thus, for this document the only valid value used in the examples is 'thttp'.

## 6. Examples

### 6.1 Example 1

Consider a URN that uses the hypothetical DUNS namespace. DUNS numbers are identifiers for approximately 30 million registered businesses around the world, assigned and maintained by Dunn and Bradstreet. The URN might look like:

urn:duns:002372413:annual-report-1997

The first step in the resolution process is to find out about the DUNS namespace. The namespace identifier[3], "duns", is extracted from the URN, prepended to urn.net, and the NAPTRs for duns.urn.net looked up. It might return records of the form:

```
duns.urn.net.
;;      order pref flags service          regexp        replacement
 IN NAPTR 100  10  "s" "dunslink+I2L+I2C" ""  dunslink.udp.isi.dandb.com.
 IN NAPTR 100  20  "s" "rcds+I2C"         ""  rcds.udp.isi.dandb.com.
 IN NAPTR 100  30  "s" "thttp+I2L+I2C+I2R" ""  thttp.tcp.isi.dandb.com.
```

The order field contains equal values, indicating that no name delegation order has to be followed. The preference field indicates that the provider would like clients to use the special dunslink protocol, followed by the RCDS protocol, and that HTTP is offered as a last resort. All the records specify the "s" flag, which will be explained momentarily. The service fields say that if we speak dunslink, we will be able to issue either the I2L or I2C requests to obtain a URL or a URC (description) of the resource. The Resource Cataloging and Distribution Service (RCDS)[7] could be used to get a URC for the resource, while HTTP could be used to get a URL, URC, or the resource itself. All the records supply the next domain name to query, none of them need to be rewritten with the aid of regular expressions.

The general case might require multiple NAPTR rewrites to locate a resolver, but eventually we will come to the "terminal NAPTR". Once we have the terminal NAPTR, our next probe into the DNS will be for a SRV or A record instead of another NAPTR. Rather than probing for a non-existent NAPTR record to terminate the loop, the flags field is used to indicate a terminal lookup. If it has a value of "s", the next lookup should be for SRV RRs, "a" denotes that A records should sought. A "p" flag is also provided to indicate that the next action is Protocol-specific, but that looking up another NAPTR will not be part of it.

Since our example RR specified the "s" flag, it was terminal. Assuming our client does not know the dunslink protocol, our next

action is to lookup SRV RRs for rcds.udp.isi.dandb.com, which will tell us hosts that can provide the necessary resolution service. That lookup might return:

```
   ;;                            Pref Weight Port Target
   _rcds._udp.isi.dandb.com IN SRV 0    0    1000 defduns.isi.dandb.com.
                            IN SRV 0    0    1000 dbmirror.com.au.
                            IN SRV 0    0    1000 ukmirror.com.uk.
```

telling us three hosts that could actually do the resolution, and giving us the port we should use to talk to their RCDS server. (The reader is referred to the SRV specification[4] for the interpretation of the fields above).

There is opportunity for significant optimization here. We can return the SRV records as additional information for terminal NAPTRs (and the A records as additional information for those SRVs). While this recursive provision of additional information is not explicitly blessed in the DNS specifications, it is not forbidden, and BIND does take advantage of it [8]. This is a significant optimization. In conjunction with a long TTL for *.urn.net records, the average number of probes to DNS for resolving DUNS URNs would approach one. Therefore, DNS server implementors SHOULD provide additional information with NAPTR responses. The additional information will be either SRV or A records. If SRV records are available, their A records should be provided as recursive additional information.

Note that the example NAPTR records above are intended to represent the reply the client will see. They are not quite identical to what the domain administrator would put into the zone files. For one thing, the administrator should supply the trailing '.' character on any FQDNs.

Also note that there could have been an additional first step where the URN was resolved as a generic URI by looking up urn.uri.net. The resulting rule would have specified that the NID be extracted from the URN and 'urn.net' appended to it resulting in the new key 'duns.urn.net' which is the first step from above.

### 6.2 Example 2

Consider a URN namespace based on MIME Content-Ids. The URN might look like this:

```
urn:cid:199606121851.1@mordred.gatech.edu
```

(Note that this example is chosen for pedagogical purposes, and does not conform to the CID URL scheme.)

The first step in the resolution process is to find out about the CID namespace. The namespace identifier, cid, is extracted from the URN, prepended to urn.net, and the NAPTR for cid.urn.net looked up. It might return records of the form:

```
 cid.urn.net.
  ;;       order pref flags service        regexp           replacement
   IN NAPTR 100   10   ""  ""  "/urn:cid:.+@([^\.]+\.)(.*)$/\2/i"    .
```

We have only one NAPTR response, so ordering the responses is not a problem. The replacement field is empty, so we check the regexp field and use the pattern provided there. We apply that regexp to the entire URN to see if it matches, which it does. The \2 part of the substitution expression returns the string "gatech.edu". Since the flags field does not contain "s" or "a", the lookup is not terminal and our next probe to DNS is for more NAPTR records: lookup(query=NAPTR, "gatech.edu").

Note that the rule does not extract the full domain name from the CID, instead it assumes the CID comes from a host and extracts its domain. While all hosts, such as mordred, could have their very own NAPTR, maintaining those records for all the machines at a site as large as Georgia Tech would be an intolerable burden. Wildcards are not appropriate here since they only return results when there is no exactly matching names already in the system.

The record returned from the query on "gatech.edu" might look like:

```
  gatech.edu.
  ;;       order pref flags service           regexp  replacement
  IN NAPTR 100  50  "s"  "z3950+I2L+I2C"     ""
_z3950._tcp.gatech.edu.
  IN NAPTR 100  50  "s"  "rcds+I2C"          ""    _rcds._udp.gatech.edu.
  IN NAPTR 100  50  "s"  "thttp+I2L+I2C+I2R" ""
_thttp._tcp.gatech.edu.
```

Continuing with our example, we note that the values of the order and preference fields are equal in all records, so the client is free to pick any record. The flags field tells us that these are the last NAPTR patterns we should see, and after the rewrite (a simple replacement in this case) we should look up SRV records to get information on the hosts that can provide the necessary service.

Assuming we prefer the Z39.50 protocol, our lookup might return:

```
   ;;                        Pref Weight   Port Target
   _z3950._tcp.gatech.edu IN SRV 0    0      1000 z3950.gatech.edu.
                          IN SRV 0    0      1000 z3950.cc.gatech.edu.
                          IN SRV 0    0      1000 z3950.uga.edu.
```

telling us three hosts that could actually do the resolution, and

giving us the port we should use to talk to their Z39.50 server.

Recall that the regular expression used \2 to extract a domain name from the CID, and \. for matching the literal '.' characters seperating the domain name components. Since '\' is the escape character, literal occurances of a backslash must be escaped by another backslash. For the case of the cid.urn.net record above, the regular expression entered into the zone file should be "/urn:cid:.+@([^\\.]+\\.)(.*)$/\\2/i". When the client code actually receives the record, the pattern will have been converted to "/urn:cid:.+@([^\.]+\.)(.*)$/\2/i".

### 6.3 Example 3

Even if URN systems were in place now, there would still be a tremendous number of URLs. It should be possible to develop a URN resolution system that can also provide location independence for those URLs. This is related to the requirement in [1] to be able to grandfather in names from other naming systems, such as ISO Formal Public Identifiers, Library of Congress Call Numbers, ISBNs, ISSNs, etc.

The NAPTR RR could also be used for URLs that have already been assigned. Assume we have the URL for a very popular piece of software that the publisher wishes to mirror at multiple sites around the world:

http://www.foo.com/software/latest-beta.exe

We extract the prefix, "http", and lookup NAPTR records for http.uri.net. This might return a record of the form:

```
http.uri.net. IN NAPTR
;;  order   pref flags service      regexp             replacement
     100     90   ""      ""   "!http://([^/:]+)!\1!i"       .
```

This expression returns everything after the first double slash and before the next slash or colon. (We use the '!' character to delimit the parts of the substitution expression. Otherwise we would have to use backslashes to escape the forward slashes, and would have a regexp in the zone file that looked like "/http:\\/\\/([^\\/:]+)/\\1/i".).

Applying this pattern to the URL extracts "www.foo.com". Looking up NAPTR records for that might return:

```
www.foo.com.
;;       order pref flags   service  regexp     replacement
 IN NAPTR 100  100  "s"   "thttp+L2R"   ""    _thttp._tcp.foo.com.
 IN NAPTR 100  100  "s"   "ftp+L2R"    ""    _ftp._tcp.foo.com.
```

Looking up SRV records for thttp.tcp.foo.com would return information on the hosts that foo.com has designated to be its mirror sites. The client can then pick one for the user.

## 7. Notes

- Registration procedures for the urn.net and uri.net DNS zones is specified in "Assignment Procedures for the URI Resolution using DNS (RFC2168)"[11]

- A client MUST process multiple NAPTR records in the order specified by the "order" field, it MUST NOT simply use the first record that provides a known protocol and service combination.

- If a record at a particular order matches the URI, but the client doesn't know the specified protocol and service, the client SHOULD continue to examine records that have the same order. The client MUST NOT consider records with a higher value of order. This is necessary to make delegation of portions of the namespace work. The order field is what lets site administrators say "all requests for URIs matching pattern x go to server 1, all others go to server 2". A match is defined as:

  1. The NAPTR provides a replacement domain name

  2. or the regular expression matches the URN

- When multiple RRs have the same "order", the client should use the value of the preference field to select the next NAPTR to consider. However, because of preferred protocols or services, estimates of network distance and bandwidth, etc. clients may use different criteria to sort the records.

- If the lookup after a rewrite fails, clients are strongly encouraged to report a failure, rather than backing up to pursue other rewrite paths.

- When a namespace is to be delegated among a set of resolvers, regexps must be used. Each regexp appears in a separate NAPTR RR. Administrators should do as little delegation as possible, because of limitations on the size of DNS responses.

- Note that SRV RRs impose additional requirements on clients.

## 8. Acknowledgments

The editors would like to thank Keith Moore for all his consultations during the development of this draft. We would also like to thank Paul Vixie for his assistance in debugging our implementation, and his answers on our questions. Finally, we would like to acknowledge our enormous intellectual debt to the participants in the Knoxville series of meetings, as well as to the participants in the URI and URN working groups.

References

[1] Sollins, S., Masinter, L., "Functional Requirements for Uniform Resource Names", RFC 1737, December 1994.

[2] Arms, B., "The URN Implementors, Uniform Resource Names: A Progress Report", D-Lib Magazine, February 1996.

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

[4] Eastlake, D., Gulbrandsen, A., "A DNS RR for specifying the location of services (DNS SRV)", January 1999.

[5] Daniel, R., "A Trivial Convention for using HTTP in URN Resolution", RFC 2169, June 1997.

[6] Mealling, M., Daniel, R., "URI Resolution Services Necessary for URN Resolution", RFC 2483, January 1999.

[7] Moore, K., Browne, S., Cox, J., Gettler, J., "Resource Cataloging and Distribution System", Technical Report CS-97-346, December 1996.

[8] Vixie, P., "Personal communication", January 1996.

[9] Sollins, K., "Architectural Principles of Uniform Resource Name Resolution", RFC 2276, January 1998.

[10] Mealling, M., Daniel, R., "The Naming Authority Pointer (NAPTR) DNS Resource Record", draft-ietf-urn-naptr-rr-03.txt, July 1999.

[11] Mealling, M., "Assignment Procedures for the URI Resolution using DNS (RFC2168)", draft-ietf-urn-urn.net-procedures-01.txt, November 1998.

Authors' Addresses

Michael Mealling
Network Solutions, Inc.
505 Huntmar Park Drive
Herndon, VA 22070
US

Phone: +1 770 935 5492
EMail: michaelm@netsol.com
URI: http://www.netsol.com

Ron Daniel
DATAFUSION, Inc.
139 Townsend Street, Ste. 100
San Francisco, CA 94107
US

Phone: +1 415 222 0100
EMail: rdaniel@datafusion.net
URI: http://www.datafusion.net

## Appendix A. IANA Considerations

The use of the "urn.net" and "uri.net" zones requires registration policies and procedures to be followed and for the operation of those DNS zones to be maintained. These policies and procedures are spelled out in a "Assignment Procedures for the URI Resolution using DNS (RFC2168)"[11]. The operation of those zones imposes operational and adminstrative responsibilities on the IANA.

The registration methods used for specifying values for the Services (both protocols and services) and Flags fields that are specific to URI resolution is for a specification to be published as an RFC and approved by the IESG.

The registration policies for URLs and URNs are also specified elsewhere and thus those impacts on the IANA are spelled out there.

## Appendix B. Security Considerations

The use of "urn.net" and "uri.net" as the registry for namespaces is subject to denial of service attacks, as well as other DNS spoofing attacks. The interactions with DNSSEC are currently being studied. It is expected that NAPTR records will be signed with SIG records once the DNSSEC work is deployed.

The rewrite rules make identifiers from other namespaces subject to the same attacks as normal domain names. Since they have not been easily resolvable before, this may or may not be considered a problem.

Regular expressions should be checked for sanity, not blindly passed to something like PERL.

This document has discussed a way of locating a resolver, but has not discussed any detail of how the communication with the resolver takes place. There are significant security considerations attached to the communication with a resolver. Those considerations are outside the scope of this document, and must be addressed by the specifications for particular resolver communication protocols.

## Appendix C. Appendix A -- Psuedo Code

For the edification of implementers, pseudocode for a client routine using NAPTRs is given below. This code is provided merely as a convience, it does not have any weight as a standard way to process NAPTR records. Also, as is the case with pseudocode, it has never been executed and may contain logical errors. You have been warned.

```
//
// findResolver(URN)
// Given a URN, find a host that can resolve it.
//
findResolver(string URN) {
  // prepend prefix to urn.net
  sprintf(key, "%s.urn.net", extractNS(URN));
  do {
    rewrite_flag = false;
    terminal = false;
    if (key has been seen) {
      quit with a loop detected error
    }
    add key to list of "seens"
    records = lookup(type=NAPTR, key); // get all NAPTR RRs for 'key'

    discard any records with an unknown value in the "flags" field.
    sort NAPTR records by "order" field and "preference" field
        (with "order" being more significant than "preference").
    n_naptrs = number of NAPTR records in response.
    curr_order = records[0].order;
    max_order = records[n_naptrs-1].order;

    // Process current batch of NAPTRs according to "order" field.
    for (j=0; j < n_naptrs && records[j].order <= max_order; j++) {
      if (unknown_flag) // skip this record and go to next one
         continue;
      newkey = rewrite(URN, naptr[j].replacement, naptr[j].regexp);
      if (!newkey) // Skip to next record if the rewrite didn't
         match continue;
      // We did do a rewrite, shrink max_order to current value
      // so that delegation works properly
      max_order = naptr[j].order;
      // Will we know what to do with the protocol and services
      // specified in the NAPTR? If not, try next record.
      if(!isKnownProto(naptr[j].services)) {
        continue;
      }
      if(!isKnownService(naptr[j].services)) {
        continue;
      }
```

```
            // At this point we have a successful rewrite and we will
            // know how to speak the protocol and request a known
            // resolution service. Before we do the next lookup, check
            // some optimization possibilities.
            if (strcasecmp(flags, "S")
             || strcasecmp(flags, "P"))
             || strcasecmp(flags, "A")) {
               terminal = true;
               services = naptr[j].services;
               addnl = any SRV and/or A records returned as additional
                       info for naptr[j].
            }
            key = newkey;
            rewriteflag = true;
            break;
          }
        } while (rewriteflag && !terminal);

        // Did we not find our way to a resolver?
        if (!rewrite_flag) {
           report an error
           return NULL;
        }

        // Leave rest to another protocol?
        if (strcasecmp(flags, "P")) {
           return key as host to talk to;
        }

        // If not, keep plugging
        if (!addnl) { // No SRVs came in as additional info, look them up
          srvs = lookup(type=SRV, key);
        }

        sort SRV records by preference, weight, ...
        foreach (SRV record) { // in order of preference
          try contacting srv[j].target using the protocol and one of the
              resolution service requests from the "services" field of the
              last NAPTR record.
          if (successful)
            return (target, protocol, service);
            // Actually we would probably return a result, but this
            // code was supposed to just tell us a good host to talk to.
          }
          die with an "unable to find a host" error;
      }
```

Full Copyright Statement