Network Working Group
Request for Comments: 3402
Obsoletes: 2915, 2168
Category: Standards Track

M. Mealling
Verisign

October 2002

# Dynamic Delegation Discovery System (DDDS) Part Two: The Algorithm

## Status of this Memo



## Copyright Notice



## Abstract

This document describes the Dynamic Delegation Discovery System (DDDS) algorithm for applying dynamically retrieved string transformation rules to an application-unique string. Well-formed transformation rules will reflect the delegation of management of information associated with the string. This document is also part of a series that is completely specified in "Dynamic Delegation Discovery System (DDDS) Part One: The Comprehensive DDDS" (RFC 3401). It is very important to note that it is impossible to read and understand any document in this series without reading the others.

Table of Contents



## 1. Introduction

The Dynamic Delegation Discovery System (DDDS) is used to implement lazy binding of strings to data, in order to support dynamically configured delegation systems. The DDDS functions by mapping some unique string to data stored within a DDDS Database by iteratively applying string transformation rules until a terminal condition is reached.

This document describes the general DDDS algorithm, not any particular application or usage scenario. The entire series of documents is specified in "Dynamic Delegation Discovery System (DDDS) Part One: The Comprehensive DDDS" (RFC 3401) [1]. It is very important to note that it is impossible to read and understand a single document in that series without reading the related documents.

The DDDS's history is an evolution from work done by the Uniform Resource Name Working Group. When Uniform Resource Names (URNs) [6] were originally formulated there was the desire to locate an authoritative server for a URN that (by design) contained no information about network locations. A system was formulated that could use a database of rules that could be applied to a URN to find out information about specific chunks of syntax. This system was originally called the Resolver Discovery Service (RDS) [7] and only applied to URNs.

Over time other systems began to apply this same algorithm and infrastructure to other, non-URN related, systems (see Section 6 for examples of other ways of using the DDDS). This caused some of the underlying assumptions to change and need clarification. These documents are an update of those original URN specifications in order to allow new applications and rule databases to be developed in a standardized manner.

This document obsoletes RFC 2168 [11] and RFC 2915 [9] as well as updates RFC 2276 [7].

## 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.

Application Unique String
: A string that is the initial input to a DDDS application. The lexical structure of this string must imply a unique delegation path, which is analyzed and traced by the repeated selection and application of Rewrite Rules.

Rewrite Rule
: A rule that is applied to an Application Unique String to produce either a new key to select a new rewrite rule from the rule database, or a final result string that is returned to the calling application. Also simply known as a Rule.

First Well Known Rule
: This is a rewrite rule that is defined by the application and not actually in the Rule Database. It is used to produce the first valid key.

Terminal Rule
: A Rewrite Rule that, when used, yields a string that is the final result of the DDDS process, rather than another database key.

Application
: A set of protocols and specifications that specify actual values for the various generalized parts of the DDDS algorithm. An Application must define the syntax and semantics of the Application Unique String, the First Well Known Rule, and one or more Databases that are valid for the Application.

Rule Database
: Any store of Rules such that a unique key can identify a set of Rules that specify the delegation step used when that particular Key is used.

Services
: A common rule database may be used to associate different services with a given Application Unique String; e.g., different protocol functions, different operational characteristics, geographic segregation, backwards compatibility, etc. Possible service differences might be message receiving services for email/fax/voicemail, load balancing over web servers, selection of a nearby mirror server, cost vs performance trade-offs, etc. These Services are included as part of a Rule to allow the Application to make branching decisions based on the applicability of one branch or the other from a Service standpoint.

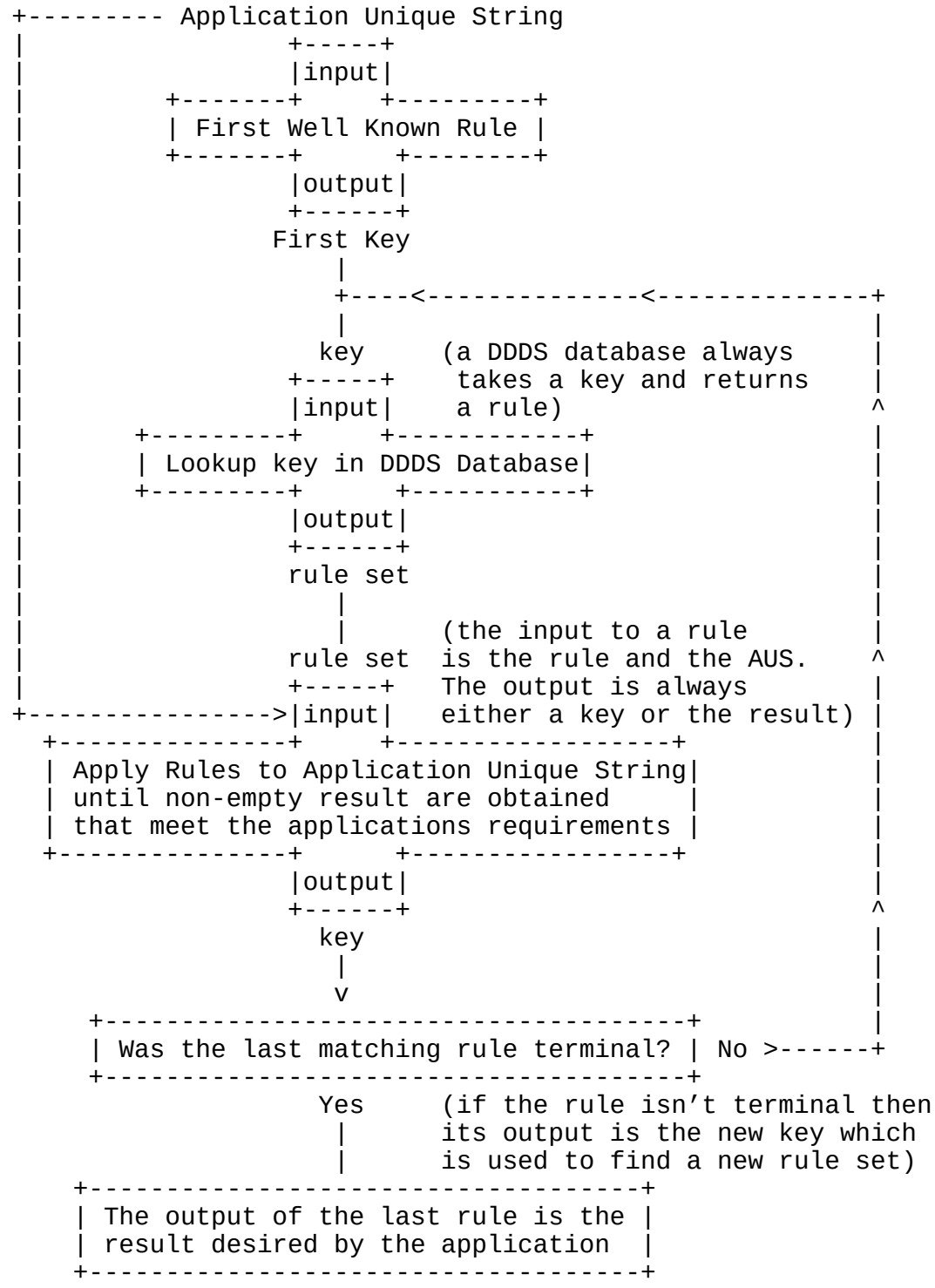
Flags
: Most Applications will require a way for a Rule to signal to the Application that some Rules provide particular outcomes that others do not; e.g., different output formats, extensibility mechanisms, terminal rule signaling, etc. Most Databases will define a Flags field that an Application can use to encode various values that express these signals.

## 3. The Algorithm

The DDDS algorithm is based on the concept of Rewrite Rules. These rules are collected into a DDDS Rule Database, and accessed by given unique keys. A given Rule, when applied to an Application Unique String, transforms that String into new Key that can be used to retrieve a new Rule from the Rule Database. This new rule is then reapplied to the original Application Unique String and the cycle repeats itself until a terminating condition is reached. An Application MUST NOT apply a Rule to the output of a previous Rule. All Rewrite Rules for all Applications must ALWAYS apply to the exact same Application Unique String that the algorithm started with.

It is a fundamental assumption that the Application Unique String has some kind of regular, lexical structure that the rules can be applied to. It is an assumption of the DDDS that the lexical element used to make a delegation decision is simple enough to be contained within the Application Unique String itself. The DDDS does not solve the case where a delegation decision is made using knowledge contained outside the AUS and the Rule (time of day, financial transactions, rights management, etc.).

Diagrammatically the algorithm looks like this:



## 3.1 Components of a Rule

A Rule is made up of 4 pieces of information:

A Priority
: Simply a number used to show which of two otherwise equal rules may have precedence. This allows the database to express rules that may offer roughly the same results but one delegation path may be faster, better, cheaper than the other.

A Set of Flags
: Flags are used to specify attributes of the rule that determine if this rule is the last one to be applied. The last rule is called the terminal rule and its output should be the intended result for the application. Flags are unique across Applications. An Application may specify that it is using a flag defined by yet another Application but it must use that other Application's definition. One Application cannot redefine a Flag used by another Application. This may mean that a registry of Flags will be needed in the future but at this time it is not a requirement.

A Description of Services
: Services are used to specify semantic attributes of a particular delegation branch. There are many cases where two delegation branches are identical except that one delegates down to a result that provides one set of features while another provides some other set. Features may include operational issues such as load balancing, geographically based traffic segregation, degraded but backwardly compatible functions for older clients, etc. For example, two rules may equally apply to a specific delegation decision for a string. One rule can lead to a terminal rule that produces information for use in high availability environments while another may lead to an archival service that may be slower but is more stable over long periods of time.

A Substitution Expression
: This is the actual string modification part of the rule. It is a combination of a POSIX Extended Regular Expression [8] and a replacement string similar to Unix sed-style substitution expression.

## 3.2 Substitution Expression Syntax

The character set(s) that the substitution expression is in and can act on are dependent both on the Application and on the Database being used. An Application must define what the allowed character sets are for the Application Unique String. A DDDS Database specification must define what character sets are required for

producing its keys and for how the substitution expression itself is encoded. The grammar-required characters below only have meaning once a specific character set is defined for the Database and/or Application.

The syntax of the Substitution Expression part of the rule is a sed-style substitution expression. True sed-style substitution expressions are not appropriate for use in this application for a variety of reasons, therefore the contents of the regexp field MUST follow this grammar:

```
subst-expr   = delim-char  ere  delim-char  repl  delim-char  *flags
delim-char   = "/" / "!" / <Any octet not in 'POS-DIGIT' or 'flags'>
                    ; All occurrences of a delim_char in a subst_expr
                    ; must be the same character.>
ere          = <POSIX Extended Regular Expression>
repl         = *(string / backref)
string       = *(anychar / escapeddelim)
anychar      = <any character other than delim-char>
escapeddelim = "\" delim-char
backref      = "\" POS-DIGIT
flags        = "i"
POS-DIGIT    = "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"
```

The result of applying the substitution expression to the String MUST result in a key which obeys the rules of the Database (unless of course it is a Terminal Rule in which case the output follows the rules of the application). Since it is possible for the regular expression to be improperly specified, such that a non-conforming key can be constructed, client software SHOULD verify that the result is a legal database key before using it.

Backref expressions in the repl portion of the substitution expression are replaced by the (possibly empty) string of characters enclosed by '(' and ')' in the ERE portion of the substitution expression. N is a single digit from 1 through 9, inclusive. It specifies the N'th backref expression, the one that begins with the N'th '(' and continues to the matching ')'. For example, the ERE

(A(B(C)DE)(F)G)

has backref expressions:

```
\1  = ABCDEFG
\2  = BCDE
\3  = C
\4  = F
\5..\9  = error - no matching subexpression
```

The "i" flag indicates that the ERE matching SHALL be performed in a case-insensitive fashion. Furthermore, any backref replacements MAY be normalized to lower case when the "i" flag is given. This flag has meaning only when both the Application and Database define a character set where case insensitivity is valid.

The first character in the substitution expression shall be used as the character that delimits the components of the substitution expression. There must be exactly three non-escaped occurrences of the delimiter character in a substitution expression. Since escaped occurrences of the delimiter character will be interpreted as occurrences of that character, digits MUST NOT be used as delimiters. Backrefs would be confused with literal digits were this allowed. Similarly, if flags are specified in the substitution expression, the delimiter character must not also be a flag character.

## 3.3 The Complete Algorithm

The following is the exact DDDS algorithm:

1. The First Well Known Rule is applied to the Application Unique String which produces a Key.

2. The Application asks the Database for the ordered set of Rules that are bound to that Key (see NOTE below on order details).

3. The Substitution Expression for each Rule in the list is applied, in order, to the Application Unique String until a non-empty string is produced. The position in the list is noted and the Rule that produced the non-empty string is used for the next step. If the next step rejects this rule and returns to this step then the Substitution Expression application process continues at the point where it left off. If the list is exhausted without a valid match then the application is notified that no valid output was available.

4. If the Service description of the rule does not meet the client's requirements, go back to step 3 and continue through the already retrieved list of rules. If it does match the client's requirements then this Rule is used for the next step. If and only if the client is capable of handling it and if it is deemed safe to do so by the Application's specification, the client may make a note of the current Rule but still return to step 3 as though it had rejected it. In either case, the output of this step is one and only one Rule.

5. If the Flags part of the Rule designate that this Rule is NOT Terminal, go back to step 2 with the substitution result as the new Key.

6. Notify the Application that the process has finished and provide the Application with the Flags and Services part of the Rule along with the output of the last Substitution Expression.

NOTE 1: In some applications and/or databases the result set can express the case where two or more Rules are considered equal. These Rules are treated as the same Rule, each one possibly having a Priority which is used to communicate a preference for otherwise equivalent Rules. This allows for Rules to act as fallbacks for others. It should be noted that this is a real Preference, not a load balancing mechanism. Applications should define the difference carefully.

NOTE 2: Databases may or may not have rules that determine when and how records within that database expire (expiration dates, times to live, etc.). These expiration mechanisms must be adhered to in all cases. Specifically, since the expiration of a databases record could cause a new Rule to be retrieved that is inconsistent with previous Rules, while in the algorithm any attempts to optimize the process by falling back to previous keys and Rules MUST ensure that no previously retrieved Rule has expired. If a Rule has expired then the application MUST start over at Step 1.

## 4. Specifying an Application

In order for this algorithm to have any usefulness, a specification must be written describing an application and one or more databases. In order to specify an application the following pieces of information are required:

Application Unique String:
: This is the only string that the rewrite rules will apply to. The string must have some regular structure and be unique within the application such that anyone applying Rules taken from the same Database will end up with the same Keys. For example, the URI Resolution application defines the Application Unique String to be a URI.

  No application is allowed to define an Application Unique String such that the Key obtained by a rewrite rule is treated as the Application Unique String for input to a new rule. This leads to sendmail style rewrite rules which are fragile and error prone. The one single exception to this is when an Application defines some flag or state where the rules for that application are

suspended and a new DDDS Application or some other arbitrary set of rules take over. If this is the case then, by definition, none of these rules apply. One such case can be found in the URI Resolution application which defines the 'p' flag which states that the next step is 'protocol specific' and thus outside of the scope of DDDS.

First Well Known Rule:
This is the first rule that, when applied to the Application Unique String, produces the first valid Key. It can be expressed in the same form as a Rule or it can be something more complex. For example, the URI Resolution application might specify that the rule is that the sequence of characters in the URI up to but not including the first colon (the URI scheme) is the first Key.

Valid Databases:
The application can define which Databases are valid. For each Database the Application must define how the First Well Known Rule's output (the first Key) is turned into something that is valid for that Database. For example, the URI Resolution application could use the Domain Name System (DNS) as a Database. The operation for turning this first Key into something that was valid for the database would be to to turn it into some DNS-valid domain-name. Additionally, for each Database an Application defines, it must also specify what the valid character sets are that will produce the correct Keys. In the URI Resolution example shown here, the character set of a URI is 7 bit ASCII which matches fairly well with DNS's 8 bit limitation on characters in its zone files.

Expected Output:
The Application must define what the expected output of the Terminal Rule should be. For example, the URI Resolution application is concerned with finding servers that contain authoritative data about a given URI. Thus the output of the terminal rule would be information (hosts, ports, protocols, etc.) that would be used to contact that authoritative server.

In the past there has been some confusion concerning load balancing and the use of the DDDS 'Priority'. Applications should be aware that the Priority of a given rule is just that: a way of specifying that one rule is "better, faster, cheaper" than another. If an application needs some method of allowing a client to load balance between servers (i.e., weighted random selection, etc.) then it should do so outside the DDDS algorithm. For example, Applications that make use of the DNS Database may use the SRV record as a way of signifying that a particular service is actually handled by several hosts cooperating with each other. The difference being that load

balancing is done between hosts that are identical to each other where as DDDS is concerned with delegation paths that have some particular feature set or administrative domain.

## 5. Specifying A Database

Additionally, any Application must have at least one corresponding Database from which to retrieve the Rules. It is important to note that a given Database may be used by more than one Application. If this is the case, each rule must be use some combination of its Services and/or substitution expression to match only those Application Unique Strings for which it is valid.

A Database specification must include the following pieces of information:

General Specification:
: The Database must have a general specification. This can reference other standards (SQL, DNS, etc.) or it can fully specify a novel database system. This specification MUST be clear as to what allowed character sets exist in order to know in which character set the Keys and Rules are encoded.

Lookup Procedure:
: This specifies how a query is formulated and submitted to the database. In the case of databases that are used for other purposes (such as DNS), the specification must be clear as to how a query is formulated specifically for the database to be a DDDS database. For example, a DNS based Database must specify which Resource Records or Query Types are used.

Key Format:
: If any operations are needed in order to turn a Key into something that is valid for the database then these must be clearly defined. For example, in the case of a DNS database, the Keys must be constructed as valid domain-names.

Rule Format:
: The specification for the output format of a rule.

Rule Insertion Procedure:
: A specification for how a Rule is inserted into the database. This can include policy statements about whether or not a Rule is allowed to be added.

Rule Collision Avoidance:
Since a Database may be used by multiple Applications (ENUM and URI Resolution for example), the specification must be clear about how rule collisions will be avoided. There are usually two methods for handling this: 1) disallow one key from being valid in two different Applications; 2) if 1 isn't possible then write the substitution expression such that the regular expression part contains enough of the Application Unique String as part of its match to differentiate between the two Applications.

## 6. Examples

The examples given here are for pedagogical purposes only. They are specifically taken from ficticious applications that have not been specified in any published document.

### 6.1 An Automobile Parts Identification System

In this example imagine a system setup where all automobile manufacturers come together and create a standardized part numbering system for the various parts (nuts, bolts, frames, instruments, etc.) that make up the automobile manufacturing and repair process. The problem with such a system is that the auto industry is a very distributed system where parts are built by various third parties distributed around the world. In order to find information about a given part a system must be able to find out who makes that part and contact them about it.

To facilitate this distributed system the identification number assigned to a part is assigned hierarchically such that the first 5 digits make up a parts manufacturer ID number. The next 3 digits are an auto line identifier (Ford, Toyota, etc.). The rest of the digits are assigned by the parts manufacturer according to rules that the manufacturer decides.

The auto industry decides to use the DDDS to create a distributed information retrieval system that routes queries to the actual owner of the data. The industry specifies a database and a query syntax for retrieving rewrite rules (the APIDA Network) and then specifies the Auto Parts Identification DDDS Application (APIDA).

The APIDA specification would define the following:

- Application Unique String: the part number.
- First Well Known Rule: take the first 5 digits (the manufacturers ID number) and use that as the Key.

- Valid Databases: The APIDA Network.

- Expected Output: EDIFAC information about the part.

The APIDA Network Database specification would define the following:

- General Specification: a network of EDI enabled databases and services that, when given a subcomponent of a part number will return an XML encoded rewrite rule.

- Lookup Procedure: following normal APIDA Network protocols, ask the network for a rewrite rule for the Key.

- Key Format: no conversion is required.

- Rule Format: see APIDA Network documentation for the XML DTD.

- Rule Insertion Procedure: determined by the authority that has control over each section of the part number. I.e., in order to get a manufacturer ID you must be a member of the Auto Parts Manufacturers Association.

In order to illustrate how the system would work, imagine the part number "4747301AB7D". The system would take the first 5 digits, '47473' and ask the network for that Rewrite Rule. This Rule would be provided by the parts manufacturers database and would allow the manufacturer to either further sub-delegate the space or point the querier directly at the EDIFAC information in the system.

In this example let's suppose that the manufacturer returns a Rule that states that the next 3 digits should be used as part of a query to their service in order to find a new Rule. This new Rule would allow the parts manufacturer to further delegate the query to their parts factories for each auto line. In our example part number the number '01A' denotes the Toyota line of cars. The Rule that the manufacturer returns further delegates the query to a supply house in Japan. This rule also denotes that this Rule is terminal and thus the result of this last query will be the actual information about the part.

### 6.2 A Document Identification Service

This example is very similar to the last since the documents in this system can simply be thought of as the auto part in the last example. The difference here is that the information about the document is kept very close to the author (usually on their desktop). Thus there is the probability that the number of delegations can be very deep. Also, in order to keep from having a large flat space of authors, the authors are organized by organizations and departments.

Let’s suppose that the Application Unique String in this example looks like the following:

<organization>-<department>-<author>:<project>-<bookcase>-<book>

The Application specification would look like this:

- Application Unique String: the Document ID string given above.
- First Well Known Rule: the characters up to but not including the first ’-’ is treated as the first Key.
- Valid Databases: the DIS LDAP Directory.
- Expected Output: a record from an LDAP server containing bibliographic information about the document in XML.

The Database specification for the DIS LDAP Directory would look like this:

- General Specification: the Database uses the LDAP directory service. Each LDAP server has a record that contains the Rewrite Rule. Rules refer to other LDAP servers using the LDAP URL scheme.
- Lookup Procedure: using standard LDAP queries, the client asks the LDAP server for information about the Key.
- Key Format: no conversion is necessary.
- Rule Format: See the LDAP Rewrite Rule specification.
- Rule Insertion Procedure: See the procedures published by the entity that has authority over that section of the DIS tree. The first section, the organization, is owned by the DIS Agency.

In this example, the first lookup is for the organization's Rule. At that point the organization may point the client directly at some large, organization wide database that contains the expected output. Other organizations may decentralize this process so that Rules end up delegating the query all the way down to the authors document management environment of choice.

## 7. Security Considerations

This document simply defines the DDDS algorithm and thus, by itself, does not imply any security issues. It is when this algorithm is coupled with a Database and an Application that security considerations can be known well enough to enumerate them beyond simply saying that dynamic delegation points are a possible point of attack.

## 8. IANA Considerations

This document does not create any requirements on the IANA. Database and Application specifications may have considerable requirements but they cannot be enumerated here.

## References

[1] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part One: The Comprehensive DDDS", RFC 3401, October 2002.

[2] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Two: The Algorithm", RFC 3402, October 2002.

[3] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database", RFC 3403, October 2002.

[4] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Four: The Uniform Resource Identifiers (URI) Resolution Application", RFC 3404, October 2002.

[5] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Five: URI.ARPA Assignment Procedures", RFC 3405, October 2002.

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

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

[8] The Institute of Electrical and Electronics Engineers, "IEEE Standard for Information Technology - Portable Operating System Interface (POSIX) - Part 2: Shell and Utilities (Vol. 1)", IEEE Std 1003.2-1992, ISBN 1-55937-255-9, January 1993.

[9] Mealling, M. and R. Daniel, "The Naming Authority Pointer (NAPTR) DNS Resource Record", RFC 2915, August 2000.

[10] Faltstrom, P., "E.164 number and DNS", RFC 2916, September 2000.

[11] Daniel, R. and M. Mealling, "Resolution of Uniform Resource Identifiers using the Domain Name System", RFC 2168, June 1997.

## Author's Address

Michael Mealling
VeriSign
21345 Ridgetop Circle
Sterling, VA 20166
US

EMail: michael@neonym.net
URI: http://www.verisignlabs.com

## Full Copyright Statement



## Acknowledgement

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