Registration Protocols Extensions
Internet-Draft
Intended status: Informational
Expires: April 23, 2017

A. Blinn
R. Carney
GoDaddy Inc.

October 20, 2016

Domain Connect API - Communications between DNS Provider and Services
draft-carney-regext-domainconnect-01

Abstract

This document provides information related to the Domain Connect API that was built to support communications between DNS Providers and Service Providers (hosting, social, email, hardware, etc.).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 23, 2017.

Table of Contents

1. Introduction

Configuring a service at a Service Provider to work with a domain is a complex task and is difficult for users.

Typically a customer will try to configure their service by entering their domain name with the Service Provider. The Service Provider then uses a number of techniques to discover the DNS Provider. This might include DNS queries to determine the registrar and/or the nameserver providing DNS.

Once the Service Provider discovers the DNS Provider, they typically give the customer instructions for proper configuration of DNS. This might include help text, screen shots, or even links to the appropriate tools.

This often presents a number of technologies or processes to the user that they may not understand. DNS record types, TTLs, Hostnames,

etc. are all confusing to many users. Instructions authored by the Service Provider may also be out of date, further confusing the issue.

The goal of the protocol presented in this RFC is to create a system where Service Providers can easily enable their applications/services to work with the domain names of their customers. This includes both discovery of the DNS Provider and subsequent modification of DNS.

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 [RFC2119].

XML is case sensitive. Unless stated otherwise, XML specifications and examples provided in this document MUST be interpreted in the character case presented in order to develop a conforming implementation.

3. Definitions

The following definitions are used in this document:

- Service Providers - refers to entities that provide products and services attached to domain names. Examples include web hosting providers (such as Wix or SquareSpace), email Service Providers (such as Microsoft or Google) and potentially even hardware manufacturers with DNS-enabled devices including home routers or automation controls (such as Linksys, Nest, and Philips).
- DNS Providers - refers to entities that provide DNS services such as registrars (e.g. GoDaddy, eNom or Tucows) or standalone DNS services (e.g. CloudFlare).
- Customer/User - refers to the end-user of these services.
- Templates/Service Templates - refers to a file that describes a set of changes to DNS and domain functionality to enable a specific service.

4. The API

The system will be implemented using simple web based interactions and standard authentication protocols, allowing for the creation and modification of DNS settings through the application of templates instead of direct manipulation of individual DNS records.

The core of this proposal is based on templates. Templates describe a service owned by a Service Provider, and contain all of the information necessary to describe the changes to the domain and to

DNS required to enable and operate/maintain a service. These changes are in the form of records/commands which map to records in DNS or other domain behavior (e.g. redirects).

The individual records/commands may be identified by a group identifier. This allows for the application of templates in different stages. For example, an email provider might first set a TXT record to verify the domain, and later set an MX record to configure email. While done separately, both changes are fundamentally part of the same service.

Templates can also contain variable portions, as often values of data in the template change based on the rules of the Service Provider (e.g. the IP address of a service).

Configuration and onboarding of templates between the DNS Provider and the Service Provider is initially seen as a manual process. The template is defined by the Service Provider and given to the DNS Provider. Future versions of this specification may allow for an independent repository of templates.

By basing the protocol on templates instead of DNS Records, several advantages are achieved. The DNS Provider has very explicit knowledge and control on the settings being changed to enable a service. The system is also more secure as templates are tightly controlled and contained.

All parties benefit by having an open standard. With more DNS Providers supporting the standard, more Service Providers are likely to adopt and vice versa.

The value to customers is simple, Domain Connect makes configuration of services much easier. Instead of editing individual DNS records, a customer simply approves the application of a template to their domain.

To attach a domain name to a service provided by a Service Provider, the customer would first enter their domain name.

Instead of relying on examination of the nameserver and mapping these to DNS Providers, DNS Provider discovery would be handled through simple records in DNS and an API. The Service Provider would query for a specific record in the zone to determine a REST endpoint, call an API, and a Domain Connect compliant DNS Provider would return information about that domain at the DNS Provider.

For the application of the changes to DNS, there are two main use cases. The first is a synchronous web flow. The second is the API when the OAuth flow is used.

4.1. Web-Based Authentication, Authorization & Template Action Flow

This flow is tailored for the Service Provider that requires a one-time synchronous change to DNS.

The user would first enter their domain name at the Service Provider.

After the Service Provider determines the DNS Provider, the Service Provider would display a link to the user indicating that they can "Connect their Domain" to the service.

After clicking the link, the user is directed to a browser window on the DNS Provider's site. This could be in place, another tab, or in a new browser window. This link would indicate the domain name being updated, the service being enabled, and any additional parameters needed to configure the service.

The user would be asked to authenticate at the DNS Provider site.

After authenticating at the DNS Provider, the DNS Provider would verify the domain name, provided by the user, is owned by the user. The DNS Provider would also verify other parameters passed in are valid and would prompt the user to give consent for making the change to DNS.

Assuming the user grants this consent, the DNS changes would be applied. Upon successful application of the DNS changes, an optional callback URL would be called at the Service Provider indicating success.

4.2. OAuth Based Authentication and Authorization Flow

The OAuth flow is tailored for the Service Provider that wishes to make changes to DNS asynchronously to the user interaction, or may wish to make multiple or additional changes to DNS over time.

The OAuth based authentication and authorization flow begins similarly to the web based synchronous flow.

However, instead of applying the DNS changes on user confirmation, OAuth access is granted to the Service Provider. An OAuth token is generated and handed back to the Service Provider.

The permission granted in the OAuth token is a right for the Service Provider to apply changes based on the template to the specific domain owned by a specific user.

The Service Provider would call an API that applies this template to the domain, including any necessary parameters along with the access token(s). As in all OAuth flows, access can be revoked by the user at any time. This would be done on the DNS Providers user experience.

If the OAuth flow is used, once a Service Provider has an OAuth token the Domain Connect API becomes available for use. The Domain Connect API is a simple REST service.

This REST service allows the application or removal of the changes in the template on a domain name. The domain name, user, and template must be authorized through the OAuth token and corresponding access token.

Additional parameters named keys are expected to be passed as name/value pairs on the query string of each API call.

4.3. DNS Provider Initiated Flows

It may be desired to expose different services available from the DNS Provider, mainly to expose interesting services that the user could attach to their domain. An example would be suggesting to a user that they might want to connect their domain to a partner Service Provider.

If the template for the service is static, it is sometimes possible to simply apply the template, and be done.

However, often the template has some dynamic elements. For this scenario, the DNS Provider need simply call a URL at the Service Provider. The Service Provider can then sign the user in, collect any necessary information, and call the normal web-based flows described above.

4.4. DNS Provider Discovery

In order to facilitate discovery of the DNS Provider given a domain name, a domain will contain a record in DNS.

This record will be a simple TXT record containing a URL used as a prefix for calling a discovery API. This record will be named domainconnect.

An example of this record would contain:

https://domainconnect.godaddy.com

As a practical matter of implementation, the DNS Provider need not contain a copy of this data in each and every zone. Instead, the DNS Provider needs simply to respond to the DNS query for the domainconnect TXT record with the appropriate data. How this is implemented is up to the DNS Provider.

Once the URL prefix is discovered, it can be used by the Service Provider to determine the additional settings for using Domain Connect on this domain at the DNS Provider. This is done by calling a REST API.

```
GET
v2/{domain}/settings
```

This will return a JSON structure containing the settings to use for Domain Connect on the domain name (passed in on the path) at the DNS Provider. This JSON structure will contain the following fields:

- providerName: The name of the DNS Provider suitable for display on the Service Provider UX.
- urlSyncUX: The URL Prefix for linking to the UX elements of Domain Connect for the synchronous flow at the DNS Provider.
- urlAsyncUX: The URL Prefix for linking to the UX elements of Domain Connect for the asynchronous flow at the DNS Provider
- urlAPI: This is the URL Prefix for the REST API for the asynchronous OAuth API.

As an example, the JSON returned by this call might contain.

```
{
"providerName": "GoDaddy",
"urlSyncUX": "https://domainconnect.godaddy.com",
"urlAsyncUX": "https://domainconnect.godaddy.com",
"urlAPI" : "https://api.domainconnect.godaddy.com"
}
```

4.5. Domain Connect Endpoints

Domain Connect contains endpoints in the form of URLs.

The first set of endpoints are for the UX that the Service Provider links to.

These are for the UX which includes the web-based flow where the user clicks on the link, and the OAuth flow where the user clicks on the link for consent.

The second set of endpoints are for the API that is called as part of the asynchronous OAuth flow via REST.

All endpoints begin with a root URL for the DNS Provider such as https://connect.dnsprovider.com/ and may also include any prefix at the discretion of the DNS Provider, for example, https://connect.dnsprovider.com/api/

The root URLs for the UX endpoints and the API endpoints are returned in the JSON payload during DNS Provider discovery.

4.5.1. Web Based Flow

GET
v2/domainTemplates/providers/{providerDomain}/services/{serviceName}/apply?[properties]

This is the URL used to apply a template to a domain. This URL is embedded on the Service Provider's site to start the Domain Connect protocol.

Parameters/properties passed to this URL include:

- domain: This parameter contains the domain name being configured.
- name/value pairs: Any variable fields consumed by this template. The name portion of this API call corresponds to the variable(s) specified in the template and the value corresponds to the value that should be used when applying the template.
- requestId: This OPTIONAL parameter may contain a value that will be passed back to the calling Service Provider via the template's callback URL. A Service Provider may use this value to identify a specific call or for any other purpose.
- groupId: This OPTIONAL parameter specifies the group of changes from the template to apply. If no group is specified, all changes are applied.

An example query string is below:

GET
https://webconnect.dnsprovider.com/v2/domainTemplates/providers/coolprovider.com/services/hosting/
apply?www=192.168.42.42&m=192.168.42.43&domain=example.com

This call indicates that the Service Provider wishes to connect the domain example.com to the service using the template identified by the composite key of the provider (coolprovider.com) and the service owned by them (hosting). In this example, there are two variables in this template, "www" and "m" which both require values (in this case each requires an IP address). These variables are passed as name/value pairs.

As part of the Domain Connect flow, a callback URL will be invoked if provided.

It should also be noted that successfully getting a callback URL invoked in a flow such as this isn't 100% reliable. Requests often fail, and users may close their web browser before such a callback is invoked.

This callback URL is largely for tracking and convenience. As such the lack of reliability is likely not a factor. A Service Provider who wishes to continue any process with certainty will simply check the DNS for any applied changes as a trigger for further action.

The URL called is specified as part of the onboarding process with the service. This URL would allow for the substitution of three values:

- domain: The domain name configured with domain connect.
- requestId: The passed in requestId in the initial call.
- status: The status or results of the operation (SUCCESS, CANCELED, FAILED, ERROR).

The format of this URL provided by the Service Provider to the DNS Provider would be similar to:

http://example.com/
connectresults?domain=%domain%&request=%requestId%&status=%status%

4.5.2. OAuth Flow

Using the OAuth flow is a more advanced use case, needed by Service Providers that have more complex configurations that may require multiple steps and/or are asynchronous from the user's interaction.

Details of an OAuth implementation are beyond the scope of this specification. Instead, an overview of how OAuth fits with Domain Connect is given here.

Service providers wishing to use the OAuth flow must register as an OAuth client with the DNS Provider. This is envisioned as a manual process.

To register, the Service Provider would provide (in addition to their template) one or more callback URLs that specify where the customer will be redirected after the provider authorization. In return, the DNS Provider will give the Service Provider a client id and secret which will be used when requesting tokens as part of the OAuth process flow.

4.5.2.1. Getting an Authorization Token

```
GET
v2/domainTemplates/
providers/{providerDomain}/services/{serviceName}
```

To initiate the OAuth flow the Service Provider would link to the DNS Provider to gain consent. This endpoint is similar to the synchronous flow described above, and will handle authenticating the user and asking for the user's permission to allow the Service Provider to make the specified changes to the domain.

Upon successful authorization, the DNS Provider will direct the end user's browser to the redirect URI provided in the request, appending the authorization code as a query parameter of "code".

Upon error, the DNS Provider will direct the end user's browser to the redirect URI provided in the request, appending the error code as a query parameter "error".

The following describes the values to be included in the query string parameters for this request.

- domain: This parameter contains the domain name being configured.
- client_id: This is the client id that was provided by the DNS Provider, to the Service Provider during registration.
- redirect_uri: The location to direct the client's browser to upon successful authorization, or upon error.
- scope: This is the name of the resource that the Service Provider is requesting access to.
- response_type: OPTIONAL. If included should be the string 'code' to indicate an authorization code is being requested.
- state: OPTIONAL but recommended. This is a random, unique string passed along to prevent CSRF. It will be returned as a parameter when redirecting to the redirect_url described above.

4.5.2.2. Requesting an Access Token

POST /v2/oauth/access_token

Once authorization has been granted the Service Provider must use the Authorization Token provided to request an Access Token. The OAuth specification recommends that the Authorization Token be a short lived token, and a reasonable recommended setting is ten minutes. As such this exchange needs to be completed before that time has expired or the process will need to be repeated.

This token exchange is done via a server to server API call from the Service Provider to the DNS Provider.

The Access Token granted will also have a short-lived lifespan, also on the order of ten minutes. To get a new access token, the Refresh Token is used.

The following describes the POST parameters to be included in the request.

- code: The authorization code that was provided in the previous step when the customer accepted the authorization request, or the refresh_token for a subsequent access token.
- redirect_uri: OPTIONAL. If included, needs to be the same redirect uri provided in the previous step, simple for verification.
- grant_type: The type of code in the request. Usually the string 'authorization_code' or 'refresh_token'.
- client_id: This is the client id that was provided by the DNS Provider, to the Service Provider during registration.
- client_secret: The secret provided to the Service Provider during registration.

Upon successful token exchange, the DNS Provider will return a response with 4 properties in the body of the response.

- access_token: The access token to be used when making API requests.
- token_type: Always the string "bearer".
- expires_in: The number of seconds until the access_token expires.
- refresh_token: The token that can be used to request new access tokens when this one has expired.

4.5.2.3. Making Requests with Access Tokens

Once the Service Provider has the access token, they can call the DNS Provider's API to make change to DNS on behalf of the user.

All calls to this API pass the access token in the Authorization Header of the request to the call to the API. More details can be found in the OAuth specifications, but as an example:

```
GET /resource/1 HTTP/1.1
Host: example.com
Authorization: Bearer mF_9.B5f-4.1JqM
```

4.5.2.4. Apply Template to Domain

```
POST
v2/domainTemplates/
providers/{providerId}/services/{serviceId}/apply?[properties]
```

The primary function of the API is to apply a template to a customer domain.

While the providerId and serviceId are also implied in the authorization, these are on the path for consistency with the synchronous flows. If not matching what is in the authorization, an error is returned.

In addition, the call must accept the following parameters:

- domain: This contains the domain name being configured. It must match the domain in the authorization token.
- name/value pairs: Any variable fields consumed by this template. The name portion of this API call corresponds to the variable(s) specified in the record and the value corresponds to the value that should be used when applying the template as per the implementation notes.
- groupId: This OPTIONAL parameter specifies the group of changes in the template to apply. If omitted, all changes are applied.

An example call is below. In this example, it is contemplated that there are two variables in this template, "www" and "m" which both require values (in this case each requires an IP address). These variables are passed as name/value pairs.

```
POST
https://connect.dnsprovider.com/v2/domainTemplates/providers/coolp
rovider.com/services/hosting/
apply?www=192.168.42.42&m=192.168.42.43
```

The API must validate the access token for the Service Provider and that the domain belongs to the customer and is represented by the token being presented. With these checks passing, the template may be applied to the domain after verifying that doing so would not cause an error condition, either because of problems with required variables or the current state of the domain itself (for example, already having a conflicting template applied).

Results of this call can include information indicating success, or an error. Errors will be 400 status codes, with the following codes defined.

- Success (204): A response of an http status code of 204 indicates that call was successful and the template applied. Note that any 200 level code should be considered a success.
- Unauthorized (401,403): A response of a 401 indicates that caller is not authorized to make this call. This can be because the token was revoked, or other access issues.
- Error (404,422): This indicates something wrong with the request itself, such as bad parameters.
- Failed (409): This indicates that the call was good, and the caller authorized, but the change could not be applied due to other conditions. This might be the application of a conflicting template or a domain state that prevents updates.

4.5.2.5. Revert Template

POST
v2/domainTemplates/
providers/{providerId}/services/{serviceId}/revert?domain={domain}

This API allows the removal of a template from a customer domain using an OAuth request.

The provider and service name in the authorizatoin must match the values in the URL. So must the domain name on the query string.

This call must validate that the template requested exists and has been applied to the domain by the Service Provider or a warning must be returned that the call would have no effect. This call must validate that there is a valid authorization token for the domain passed in or an error condition must be reported.

An example query string might look like:

POST
https://connect.dnsprovider.com/v2/domainTemplates/providers/coolprovider.com/services/hosting/revert?domain=example.com

Response codes are identical to above.

4.5.2.6. Revoke Access

Like all OAuth flows, the user can revoke the access at any time using UX at the DNS Provider site. So the Service Provider needs to be aware that their access to the API may be denied.

4.6. Domain Connect Objects and Templates

This description represents the values in the template. Since onboarding of a Service Provider with a DNS Provider is initially a manually oriented process, this format is a recommendation.

There may be a repository of templates in the future.

A template is defined as a standard JSON data structure containing the following data:

- providerId: The unique identifier of the Service Provider that created this template. This is used in the URLs to identify the Service Provider. To ensure non-coordinated uniqueness, this would be the domain name of the Service Provider.
- providerName: The name of the Service Provider. This will be displayed to the user.
- templateId: The name or identifier of the template. This is used in URLs to identify the template.
- templateName: The friendly name of this service. This will be displayed to the user.
- logoUrl: A graphical logo for use in any web-based flow. This is a URL to a graphical logo sufficient for retrieval.
- description: A textual description of what this template attempts to do. This is meant to assist integrators, and therefore should not be displayed to the user.
- launchUrl: OPTIONAL. A URL suitable for a DNS Provider to call to initiate the execution of this template. This allows the flow to begin with the DNS Provider as described above.
- returnUrl: OPTIONAL. The URL to call indicating the status of the call.
- records: A list of records and/or actions for the template.

Each template record is an entry that contains a type and several optional parameters based on the value.

For all entries of a record template other than "type" and "groupId", the value can contain variables denoted by %<variable name>%. These are the values substituted at runtime when writing into DNS.

It should be noted that as a best practice, the variable should be equal to the portion of the values in the template that change as little as possible.

For example, say a Service Provider requires a CNAME of one of three values for their users: s01.example.com, s02.example.com, and s03.example.com.

The value in the template could simply contain %servercluster%, and the fully qualified string passed in. Alternatively, the value in the template could contain s%var%.example.com. By placing more fixed data into the template, the data is more constrained. And by using a generic name the values in the query string are more obscured.

Each record will contain the following elements:

- type: Describes the type of record in DNS, or the operation impacting DNS. Valid values include: A, AAAA, CNAME, MX, TXT, SRV, NS, APEXCNAME, REDIR301, or REDIR302.
- groupId: This OPTIONAL parameter identifies the group the record belongs to when applying changes.
- host: The host for A, AAAA, CNAME, TXT, and MX values. This is the hostname in DNS.
- pointsTo: The pointsTo location for A, AAAA, CNAME, MX, and APEXCNAME records.
- ttl: This is the time-to-live for the record in DNS. Valid for A, AAAA, CNAME, TXT, MX, and SRV records.
- data: This is the data for a TXT record in DNS.
- priority: This is the priority for an MX or SRV record in DNS.
- weight: This is the weight for the SRV record.
- port: This is the port for the SRV record.
- protocol: This is the protocol for the SRV record.
- service: This is the protocol for the SRV record.
- target: This is the target url for REDIR301 and REDIR302.

4.7. Implementation Notes

This template format is intended for internal use by a DNS Provider and there are no codified API endpoints for creation or modification of these objects. API endpoints do not use this object directly. Instead, API endpoints reference a template by ID and then provide key/value pairs that match any variable values in these record objects.

However, by defining a standard template format it is believed it will make it easier for Service Providers to share their provisioning across DNS Providers. Further revisions of this specification may include a repository for publishing and consuming these templates.

Implementers are responsible for data integrity and should use the record type field to validate that variable input meets the criteria for each different data type.

Certain record types may not be valid with others (e.g. a redirect and an A record), and it is up to the DNS and Service Providers to author templates appropriately. As such, a practical matter may be the redirect is valid only by itself.

Additional record types and/or extensions to the data that can be set into the template can be implemented on a per DNS Provider basis. For example, if a DNS Provider supports additional record types, these can be added to this specification and templates.

Similarly other providers may not wish to support certain record types (redirects, APEXCNAME). Should this be the case, a Service Provider depending on this functionality would not be able to operate with said DNS Provider.

Example Records: Single static host record

Consider a template for setting a single host record. The records section of the template would have a single record of type "A" and could have a value of:

```
[{
''type'': ''A'',
''host'': ''www'',
''pointsTo'': ''192.168.1.1'',
''ttl'': 600
}]
```

This would have no variable substitution and the application of this template to a domain would simply set the host name "www" to the IP address "192.168.1.1"

Example Records: Single variable host record for A

In the case of a template for setting a single host record from a variable, the template would have a single record of type "A" and could have a value of:

```
[{
''type'': ''A'',
''host'': ''@'',
''pointsTo'': ''192.168.1.%srv%'',
''ttl'': 600
}]
```

A query string with a key/value pair of

srv=8

would cause the application of this template to a domain to set the host name for the apex A record to the IP address "192.168.1.8" with a TTL of 600.

Example: Multiple variable host record for A

In the case of a template for setting a single host record from multiple variables, the template would have a single record of type "A" and could have a value of:

```
[{
''type'': ''A'',
''host'': ''%hostname1%'',
''pointsTo'': ''%hostip1%'',
''ttl'': 600
}]
```

A query string with key/value pairs of

hostname1=example&hostip1=192.168.1.3

would cause the application of this template to a domain to set the host name "example" to the IP address "192.168.1.3" with a TTL of 600.

Example: Redirect

In the case of a template for setting an HTTP redirect, the template would have a record of type "REDIRECT" and could have a value of:

```
[{
''type'': REDIR301,
''target'': %url%
}]
```

A query string with key/value pairs of

url=http://www.example-two.com.

would cause the application of this template to signal to the DNS Provider to provision URL redirection to the target URL.

Example Template JSON Format

```
{
"providerId": "example.com",
"providerName": "Example Web Hosting",
"templateId": "hosting",
"templateName": "Wordpress by example.com",
"logoUrl": "https://www.example.com/images/billthecat.jpg",
"description": "This connects your domain to our super cool web
hosting",
"returnUrl": "https://www.example.com/connectresults",
"launchURL" : "https://www.example.com/connectlaunch",
"records": [


   {
   "groupId" : "service",
   "type": "A",
   "host": "www",
   "pointsTo": "%var1%",
   "ttl": "%var2%"
   },
   {
   "groupId" : "service",
   "type": "A",
   "host": "m",
   "pointsTo": "%var3%",
   "ttl": "%var2%"
   },
   {
   "groupId" : "service",
   "type": "CNAME",
   "host": "webmail",
   "pointsTo": "%var4%",
   "ttl": "%var2%"
   },
   {
   "groupId" : "verification",
   "type": "TXT",
   "host": "example",
   "pointsTo": "%var5%",
   "ttl": "%var2%"
   }
]
}
```

4.8. Operational and Implementation Considerations

From a DNS Provider standpoint, it is envisioned that the user has appropriate warnings and checks in place to prevent accidental destruction of other records in DNS when applying a template or making manual changes in DNS.

For example, if the application of a template through the web based flow would interfere with previously set DNS records (either through another template or manual settings), it is envisioned that the user would be asked to confirm the clearing of the previously set template. If it would interfere with DNS records accessible through a previously issued OAuth flow, the provider could revoke the previously issued token.

Similarly, when granting an OAuth token that interferes with a previously issued OAuth token, access to the old token could automatically be revoked.

By doing so, this minimizes if not eliminates the case where an OAuth token cannot be applied due to conflicting templates or records existing on the domain.

Manual changes to DNS through the DNS Provider could have appropriate warnings in place to prevent unwanted changes; with overrides being possible removing conflicting templates.

The behavior of these interactions is left to the sophistication of the DNS Provider.

Variables in templates that are hard-coded host names are the responsibility of the DNS Provider to protect. That is, DNS Providers are responsible for ensuring that host names do not interfere with known values (such as m. or www. or mail.) or internal names that provide critical functionality that is outside the scope of this specification.

5. IANA Considerations

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [RFC3688]. The following URI assignment is requested of IANA:

URI: ietf:params:xml:ns:validate-1.0

Registrant Contact: See the "Author's Address" section of this document.

6. Acknowledgements

The authors wish to thank the following persons for their feedback and suggestions:

- Chris Ambler of GoDaddy Inc.
- Jody Kolker of GoDaddy Inc.

7. Change History

7.1. Change from 00 to 01

Minor edits and clarifications found during implementation.

8. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

Authors' Addresses

Arnold Blinn
GoDaddy Inc.
14455 N. Hayden Rd. #219
Scottsdale, AZ 85260
US

Email: arnoldb@godaddy.com
URI: http://www.godaddy.com

Roger Carney
GoDaddy Inc.
14455 N. Hayden Rd. #219
Scottsdale, AZ 85260
US

Email: rcarney@godaddy.com
URI: http://www.godaddy.com