Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 5 March 2026

P. Kowalik
DENIC eG
A. Blinn

J. Kolker
GoDaddy Inc.
S. Kerola
Cloudflare, Inc.
1 September 2025

Domain Connect Protocol - DNS provisioning between Services and DNS Providers
draft-kowalik-domainconnect-02

Abstract

This document provides specification of the Domain Connect Protocol that was built to support DNS configuration provisioning between Service Providers (hosting, social, email, hardware, etc.) and DNS Providers.

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 https://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 5 March 2026.

Copyright Notice

Table of Contents

1. Acknowledgements

The authors wish to thank the following persons for their feedback and suggestions as well as for the previous work on the standard:

* Roger Carney of GoDaddy Inc.

* Chris Ambler of GoDaddy Inc.

* Darrel Miller

* Peter Thomassen

* Paul Hoffmann

* Arnt Gulbrandsen

2. Introduction

Connecting a domain name to a service should be a simple and straightforward process. However, historically, users have faced a complex and often frustrating task involving manual DNS configuration. Traditional methods are unreliable, require deep technical knowledge of DNS, and result in outdated and confusing instructions from Service Providers. This leads to user frustration, support overhead, and abandoned setups.

To address these challenges, Domain Connect offers a streamlined and automated solution. It empowers Service Providers to easily enable their services to work with user domains, simplifying both DNS provider discovery and DNS configuration. By abstracting away the complexities of manual DNS management through user-friendly web interactions, standard authentication, and template-based configurations, Domain Connect significantly improves the user experience.

3. Use Cases

3.1. Primary Use Cases

The following use cases illustrate the wide range of applications where Domain Connect simplifies and automates DNS configuration, from basic service onboarding to complex, dynamic DNS management scenarios.

* *SaaS Provider with One-Off DNS Configuration:* A Software as a Service (SaaS) Provider offering functionality with an option to assign own domain name, such as web hosting or email, can utilize Domain Connect to streamline the process of configuring DNS records for their customers. This automation eliminates the need for manual configuration and simplifies the onboarding experience for users.

* *SaaS Provider with Multi-Step DNS Configuration:* Some SaaS Providers may require a multi-step DNS configuration process, potentially involving asynchronous operations. For example, a service might require initial verification of domain ownership through a TXT record, followed by the creation of CNAME records for different subdomains. Domain Connect can handle such scenarios by utilizing its asynchronous flow. This allows the Service Provider to obtain user consent and apply the necessary DNS changes in multiple steps, even if the user is not actively present during the entire process.

* *On-Premise Service with Publicly Accessible DNS Service:* An on-premise service, such as a local network device or server, can also benefit from Domain Connect if it utilizes a publicly accessible DNS service. By leveraging Domain Connect, the service can automatically update DNS records as needed, ensuring that the service remains accessible through its domain name.

* *Tool or Service with Regularly Updated DNS Entries:* A tool or service that requires regular updates to DNS entries, such as a dynamic DNS service or a DNS-based load balancer, can use Domain Connect to automate the process.

* *Packaged Software Provider:* A packaged software provider, whether open-source or proprietary, can integrate Domain Connect into their installation and configuration process. This allows the software to automatically configure necessary DNS records during installation, simplifying the setup process for users. However, if the software is installed on a private network with a private DNS service, it might not be directly compatible with Domain Connect, unless the DNS service provides Domain Connect endpoints accessible to the installation process.

3.2. Use Cases out of scope

While Domain Connect offers significant advantages in automating DNS configuration, it’s important to recognize scenarios where it might not be the ideal solution:

* *Automation or CI/CD Pipelines:* Domain Connect is primarily designed for user-driven DNS configuration, where an end user grants consent and applies changes. Automating this process within CI/CD pipelines or other automated workflows can be challenging, as it requires obtaining and securely storing OAuth tokens beforehand. However, if authorisation tokens are pre-obtained from a user-driven setup process, Domain Connect can be also integrated into automation workflows.

* *Private/Enterprise DNS with Public SaaS Providers:* Domain Connect relies on public DNS records and endpoints to facilitate discovery and configuration. If a private or enterprise DNS service is used, it might not be directly compatible with Domain Connect, unless the DNS service provides publicly accessible Domain Connect endpoints.

4. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The Terms like "*Registrar*", "*Authoritative server*", "*Zone*", "*Zone Apex*" or "*Sub Domain*" are used as defined in [RFC8499].

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234]. The following ABNF rules are imported from the normative references [RFC5234].

```
ALPHA          =  %x41-5A / %x61-7A   ; A-Z / a-z
DIGIT          =  %x30-39             ; 0-9
```

Service Provider An entity that offers products and services that are configured or accessed using domain names. These services typically rely on DNS for setup, discovery and/or operation. Examples include web hosting, email services, cloud platforms, and other online applications.

DNS Provider An entity that offers DNS zone hosting services. DNS Providers are responsible for hosting the DNS zone for a domain name and providing the necessary tools to manage the DNS records. DNS Provider would be an Authoritative server operator for the hosted zones, or would have a contractual relationship with the operator to manage zone distribution over DNS.

User Refers to the end-user who has means to control domain name’s DNS configuration at DNS Provider and wishes to configure it to work with a service provided by a Service Provider.

Service Template/Template A structured data format that describes a set of configurations for DNS records required by a Service Provider to configure a certain service together with metadata related to the control flow of Domain Connect protocol. A template is used as a mean of communication between Service Provider and DNS Provider.

Public Template Repository A publicly accessible repository of Service Templates, formatted in a standardized manner, intended to facilitate sharing, discovery, and reuse of service configurations. (Read more: Section 10.11).

5. Protocol design

5.1. Templates

Templates are core to Domain Connect, as they fully describe a service owned by a Service Provider and contain all of the information necessary to enable and operate/maintain the service in the form of a set of records.

The individual records in a template MAY be assigned to a group identified by a groupId. 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 delivery. While done separately, both changes are fundamentally part of the same service.

Templates MAY also contain variable portions, as often values of data in DNS change based on the implementation and/or user of the service (e.g. the IP address of a service, a user id, etc.).

The template is defined by the Service Provider and manually onboarded with the DNS Provider, according to a template definition published in the Public Repository (Section 10.11) or agreed out-of-band between the Service Provider and the DNS Provider.

5.2. Trust Model

The Domain Connect protocol relies on a robust trust model to ensure the security and reliability of DNS configuration delegation between Service Providers and DNS Providers. This model addresses the necessary trust relationships between users, Service Providers, and DNS Providers, centering on secure template utilization and DNS Provider vetting processes.

5.2.1. Trust Establishment

User trust in the DNS Provider is an essential factor. DNS Provider is a trusted party by the fact that DNS Provider has full technical access to the DNS zone already. Users depend on their DNS Providers to accurately reflect DNS record modifications initiated through Domain Connect. Consequently, DNS Providers are expected implement authorization checks to validate user permissions prior to enacting any DNS zone changes. Furthermore, DNS Providers are responsible for

translating template specifications into a human-readable format, enabling users to readily comprehend the nature and impact of proposed DNS record changes.

Users would typically place trust in Service Providers to act responsibly and only make DNS modifications that are within the scope defined by the approved template, however it shall be observed that a malicious actor may try to exploit this trust by acting as a benign Service Provider or building a fake one, therefore by default Service Provider shall not be assumed to be a trusted party in the protocol considerations.

A foundational element to close this gap is the establishment of trust between DNS Providers and Service Providers. DNS Providers verify the legitimacy and security of templates provided by Service Providers as well as they are able to verify textual information included in the template and provided to the user, such as provider name or service name. This trust is typically established through an onboarding process which may involve contractual agreements or appropriate template acceptance policies.

5.2.2. Template Security

Templates are central to the trust model, defining the permitted scope of DNS zone modifications and restricting Service Provider actions, preventing unauthorized changes beyond the template's specified parameters. The template defines the types of DNS records within the zone and together with a distinct domain name and optionally sub-domain with user consent it defines clearly the part of the zone that the Service Provider is authorized to modify via the template application. Service Providers create templates which are then onboarded by DNS Providers, either adhering to a published template definition or based on bilateral agreements. The DNS Provider has very explicit knowledge and control of the settings being changed to enable a service.

6. Protocol Flows

6.1. General information

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

Instead of relying on examination of the nameservers and mapping these to DNS Providers, DNS Provider discovery is handled through simple records in DNS and an API. The Service Provider queries for a specific record in the zone that returns a REST endpoint to initiate the protocol. When this endpoint is called, a Domain Connect compliant DNS Provider returns information about that domain and how to configure it using Domain Connect.

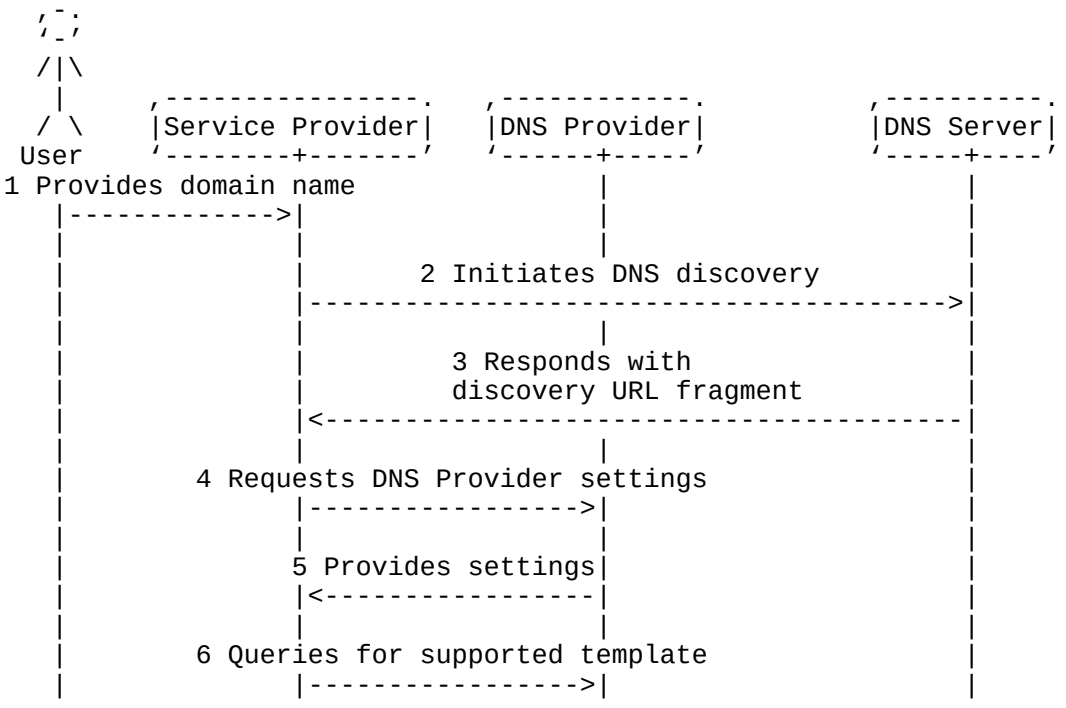
To apply the changes to DNS, there are two use cases. The first is a synchronous web flow, and the second is an asynchronous flow using OAuth and an API.

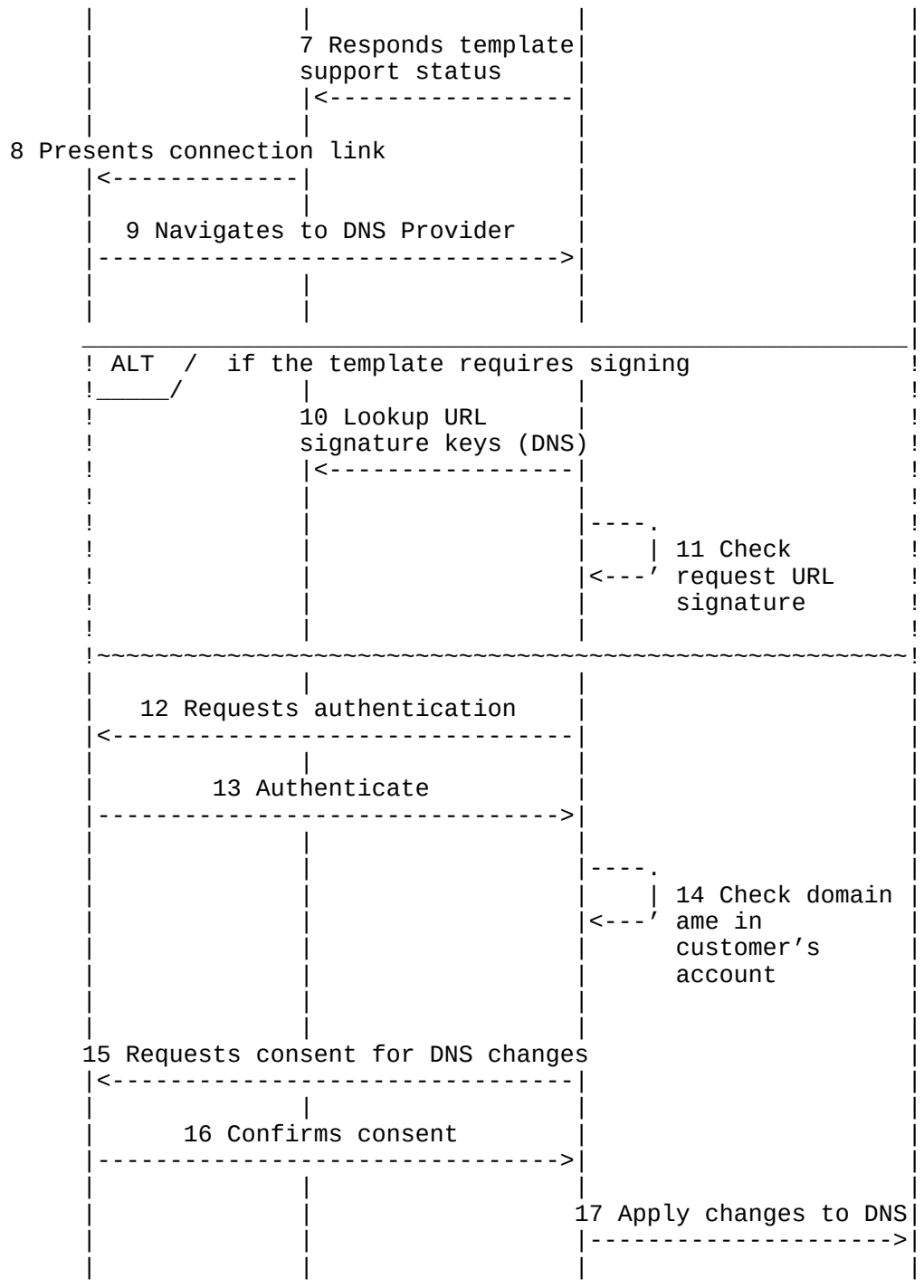
It is noted that a DNS Provider MAY choose to only implement one of the flows, however it is RECOMMENDED to implement Synchronous Flow which fulfill needs of most Service Providers.

Individual Service Providers MAY work with the synchronous flow only, the asynchronous flow only, or with both.

6.2. The Synchronous Flow

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




7 Responds template support status
8 Presents connection link
9 Navigates to DNS Provider
ALT / if the template requires signing
10 Lookup URL signature keys (DNS)
11 Check request URL signature
12 Requests authentication
13 Authenticate
14 Check domain ame in customer’s account
15 Requests consent for DNS changes
16 Confirms consent
17 Apply changes to DNS

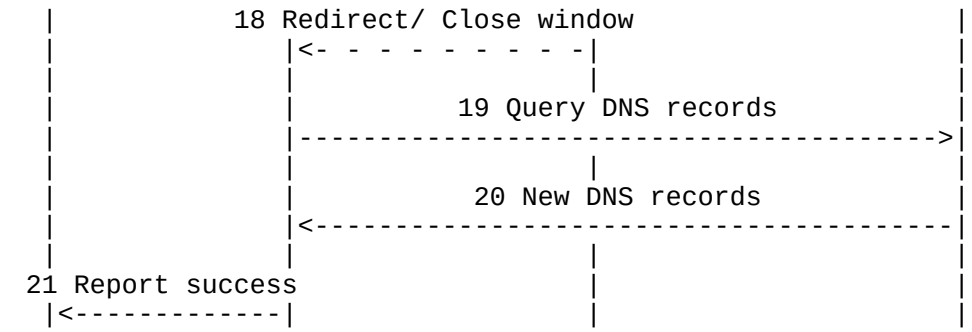


Figure 1: Sequence diagram of Synchronous Flow

Steps:

1. *User Provides Domain Name*: The user initiates the process by providing their domain name to the Service Provider.

2. *Service Provider Initiates DNS Discovery*: The Service Provider queries the DNS provider to discover the Domain Connect settings for the given domain.

3. *DNS Provider Responds with Discovery URL Fragment*: The DNS Provider responds with a URL fragment containing information where to query settings of DNS provider for a domain name.

4. *Service Provider Requests DNS Provider Settings*: The Service Provider uses the URL fragment to request the complete Domain Connect settings from the DNS Provider.

5. *DNS Provider Provides Settings*: The DNS Provider provides the settings, including information about API endpoints.

6. *Service Provider Queries for Supported Template*: The Service Provider checks if the DNS Provider supports the specific template required for the service.

7. *DNS Provider Responds with Template Support Status*: The DNS Provider confirms if they support the requested template.

8. *Service Provider Presents Connection Link*: The Service Provider presents a connection link to the user, which leads to the DNS Provider’s Domain Connect service.

9. *User Navigates to DNS Provider*: The user navigates the link and user agent is directed to the DNS Provider’s website.

10. *DNS Provider Performs URL Lookup and Signature Key Verification (if required)*: If the template requires signing, the DNS Provider looks up the URL signature keys in DNS.

11. *DNS Provider Checks Request URL Signature (if required)*: The DNS Provider verifies the signature of the request URL.

12. *Service Provider Requests Authentication*: The Service Provider requests authentication from the user.

13. *User Authenticates*: The user authenticates with the DNS Provider.

14. *DNS Provider Checks Domain Name in Customer’s Account*: The DNS Provider verifies that the user is authorized to make change to the domain’s DNS zone.

15. *DNS Provider Requests Consent for DNS Changes*: The DNS Provider asks the user for consent to apply the changes to the DNS zone.

16. *User Confirms Consent*: The user confirms their consent to the DNS changes.

17. *DNS Provider Applies Changes to DNS*: The DNS Provider applies the changes to the zone.

18. *DNS Provider Redirects or Closes Window*: The DNS Provider either redirects the user back to the Service Provider or closes the Domain Connect browser window.

19. *Service Provider Queries DNS Records*: The Service Provider queries the DNS records to verify that the changes have been applied.

20. *DNS Server Returns New DNS Records*: The DNS Server returns the updated DNS records.

21. *Service Provider Reports Success*: The Service Provider reports to the user that the domain has been successfully connected to the service.

6.3. The Asynchronous Flow

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

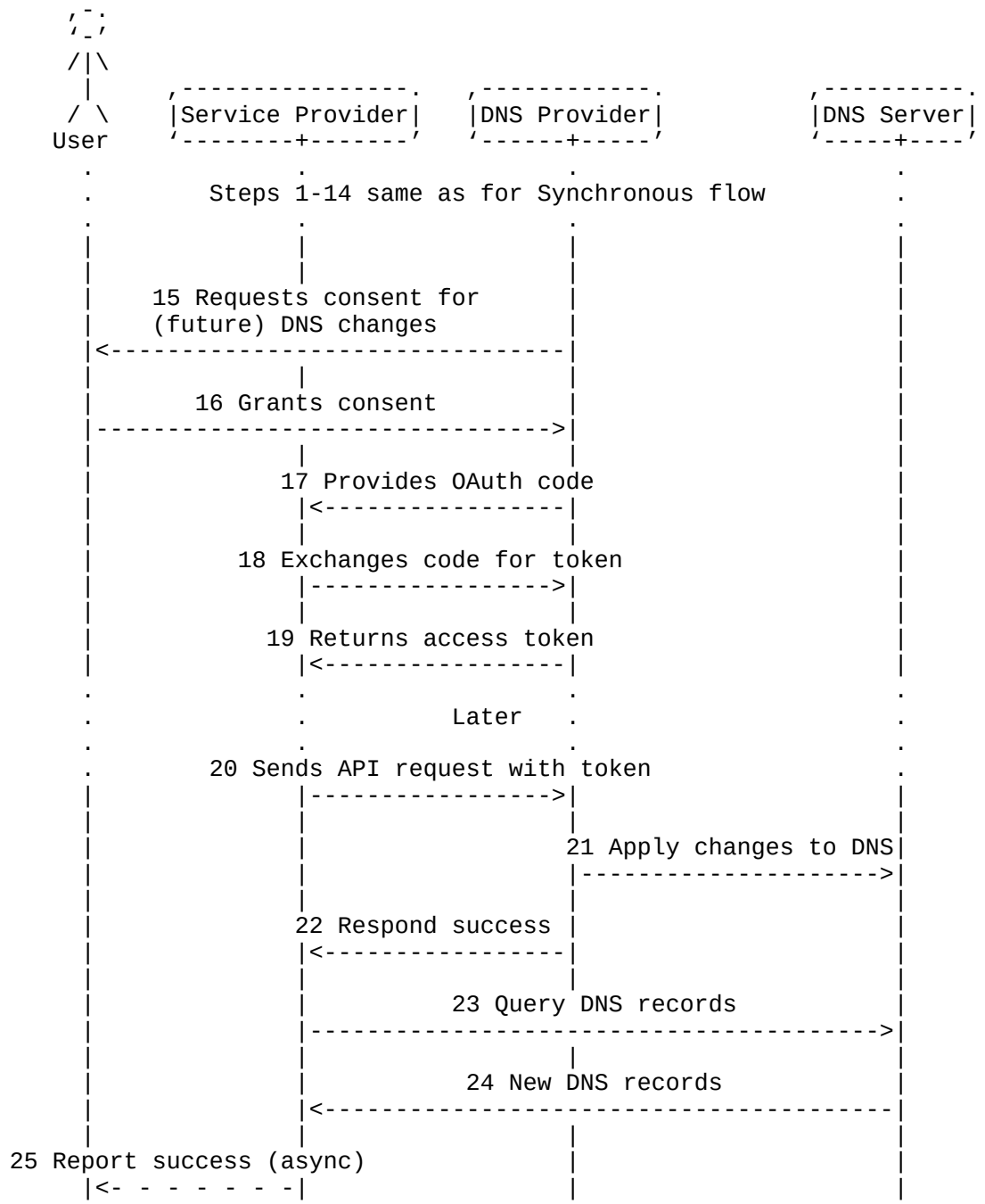


Figure 2: Sequence diagram of Asynchronous Flow

Steps:

1-14: Same as for the Synchronous Flow.

15. *DNS Provider Requests Consent for (Future) DNS Changes*: The DNS Provider asks the user for consent to allow the Service Provider to make DNS changes on their behalf in the future.

16. *User Grants Consent*: The user grants consent for future DNS changes.

17. *DNS Provider Provides OAuth Code*: The DNS Provider provides an OAuth code to the Service Provider.

18. *Service Provider Exchanges Code for Token*: The Service Provider exchanges the OAuth code for an access token.

19. *DNS Provider Returns Access Token*: The DNS Provider provides an access token to the Service Provider.

20. *Service Provider Sends API Request with Token (Later)*: At a later time, the Service Provider uses the access token to send an API request to apply the template to the domain.

21. *DNS Provider Applies Changes to DNS*: The DNS Provider applies the changes to the DNS zone.

22. *DNS Provider Responds with Success*: The DNS Provider responds to the Service Provider with success.

23. *Service Provider Queries DNS Records*: The Service Provider queries the DNS records to verify that the changes have been applied.

24. *DNS Server Returns New DNS Records*: The DNS Server returns the updated DNS records.

25. *Service Provider Reports Success (Asynchronous)*: The Service Provider reports to the user that the domain has been successfully connected to the service.

7. DNS Provider Discovery

To facilitate discovery of the DNS Provider from a domain name DNS is utilized. This is done by returning a TXT record for _domainconnect in the zone.

The record content represents an authority and path part of the settings REST API URL.

An example of the contents of this record:

domainconnect.virtucondomains.example

_domainconnect TXT record content, when prepended with https:// schema and appended with /v2 path segment, MUST form a valid URL [RFC3986]. _domainconnect TXT record MUST contain the authority part of the URL and MAY contain a path part. _domainconnect MUST not contain schema, query or fragment part of an URL.

As a practical matter of implementation, the DNS Provider may or may not contain a copy of this data in each and every zone. Instead, the DNS Provider MUST simply respond to the DNS query for the _‘domainconnect‘ TXT record with the appropriate data.

How this is implemented is up to the DNS Provider.

For example, the DNS Provider may not store the data inside a TXT record for the domain, opting instead to put a CNAME in the zone and have the TXT record in the target of the CNAME. Another DNS Provider may simply respond with the appropriate records at the DNS layer without having the data in each zone.

The URL prefix returned MUST be subsequently 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.

Normative URI template of the settings end-point per [RFC6570]:

GET

https://{+_domainconnect}/v2/{domain}/settings

_domainconnect parameter is the URL prefix returned in the _domainconnect TXT record.

This MUST 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 MUST contain the following fields unless otherwise specified.

Field	*Key*	*Type*	*Description*
Provider Id	providerId	String	(REQUIRED) Unique identifier for the DNS Provider. To ensure non-coordinated uniqueness, this SHOULD be the domain name of the DNS Provider (e.g. virtucom.example).
Provider Name	providerName	String	(REQUIRED) The name of the DNS Provider.
Provider Display Name	providerDisplayName	String	(OPTIONAL) The name of the DNS Provider that SHOULD be displayed by the Service Provider. This MAY change per domain for some DNS Providers that power multiple brands.
UX URL Prefix for Synchronous Flows	urlSyncUX	String	(OPTIONAL) The URL Prefix for linking to the UX of Domain Connect for the synchronous flow at the DNS Provider. If not returned, the DNS Provider is not supporting the synchronous flow on this domain. This URL MUST be of https schema and MUST NOT contain query or fragment part.

UX URL Prefix for Asynchronous Flows	urlAsyncUX	String	(OPTIONAL) The URL Prefix for linking to the UX elements of Domain Connect for the asynchronous flow at the DNS Provider. If not returned, the DNS Provider is not supporting the asynchronous flow on this domain. This URL MUST be of https schema and MUST NOT contain query or fragment part.
API URL Prefix	urlAPI	String	(REQUIRED) The URL Prefix for the REST API This URL MUST be of https schema and MUST NOT contain query or fragment part.
Width of Window	width	Number	(OPTIONAL) This is the desired width of the window for granting consent when navigated in a popup. Default value if not returned is 750px. The Service Providers SHOULD obey to this setting if opening a pop-up window to assure optimal display of the user interface of the DNS Proviver.
Height of Window	height	Number	(OPTIONAL) This is the desired height of the window for

			granting consent when navigated in a popup. Default value if not returned is 750px. The Service Providers SHOULD obey to this setting if opening a pop-up window to assure optimal display of the user interface of the DNS Proviver.
UX URL Control Panel	urlControlPanel	String	(OPTIONAL) This is a URL to the control panel for editing DNS at the DNS Provider. This field allows a Service Provider whose template isn’t supported at the DNS Provider to provide a direct link to perform manual edits. To allow deep links to the specific domain, this string MAY contain %domain% which MUST be replaced with the domain name when building the deep link to the control panel of a given domain. This URL MUST be of https schema and MAY contain query or fragment part.
Name Servers	nameServers	String List	(OPTIONAL) This is the list of

			nameservers desired by the DNS Provider for the zone to be authoritative. This does not indicate the authoritative nameservers; for this the registry would be queried.

Table 1: properties of the settings data structure

```
{
    "providerId": "virtucondomains.example",
    "providerName": "Virtucon Domains",
    "providerDisplayName": "Virtucon Domains",
    "urlSyncUX": "https://domainconnect.virtucondomains.example",
    "urlAsyncUX": "https://domainconnect.virtucondomains.example",
    "urlAPI": "https://api.domainconnect.virtucondomains.example",
    "width": 750,
    "height": 750,
    "urlControlPanel": "https://domaincontrolpanel.virtucondomains.ex
    ample/?domain=%domain%",
    "nameServers": ["ns01.virtucondomainsdns.example", "ns02.virtucon
    domainsdns.example"]
}
```

Discovery MUST work on the Zone Apex only. Bear in mind that zones can be delegated to other users, making this information valuable to Service Providers since DNS changes may be different for a Zone Apex vs. a Sub Domain for an individual service.

The Service Provider MUST handle the condition when a query for the _domainconnect TXT record suceeds, but a call to query for the JSON fails. This can happen if the zone is hosted with another DNS Provider, but contains an incorrect _domainconnect TXT record.

The DNS Provider MUST return a 404 HTTP error code if they do not contain the zone.

Status	Response	Description
Success	2xx	A response of an http status code of 2xx indicates that the call was successful. The response is the JSON described above.
Not Found	404	A response of a 404 indicates that the DNS Provider does not have the zone.

Table 2: HTTP status codes for the settings end-point

8. Applying Domain Connect

8.1. Endpoints

The Domain Connect endpoints returned in the JSON during discovery are in the form of URLs.

The first set of endpoints are for the UX that the Service Provider links to. These are for the synchronous flow where the user can click to grant consent and have changes applied, and for the asynchronous OAuth flow where the user can grant consent for OAuth access.

The second set of endpoints are for the REST API.

All endpoints begin with a root URL for the DNS Provider such as:

https://connect.dnsprovider.example

They MAY also include any path segment at the discretion of the DNS Provider. For example:

https://connect.dnsprovider.example/api

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

8.2. Query Supported Template

Normative URI template of the template query end-point per [RFC6570]:

GET

{+urlAPI}/v2/domainTemplates/providers/{providerId}/services
/{serviceId}

This URL is be used by the Service Provider to determine if the DNS Provider supports a specific template.

The following table describes the parameters of the URI template:

Property	Key	Description
URL API	urlAPI	(REQUIRED) Value of urlAPI from the settings endpoint.
Service Provider Id	providerId	(REQUIRED) identifier of the Service Provider of the template.
Service Id	serviceId	(REQUIRED) The name or identifier of the template.

Table 3: URI template parameters for the query supported template end-point

Returning a status of 200 without a body indicates the template is supported. The DNS Provider MAY disclose the version of the template in a JSON object with field version (see: version field (Section 9.2) or the full JSON object of deployed template.

Returning a status of 404 indicates the template is not supported.

Status	Response	Description
Success	2xx	A response of an http status code of 2xx indicates that the call was successful. The response OPTIONALLY contains the version or template.
Not Found	404	A response of a 404 indicates that the template is not supported

Table 4: https status codes for the Query Supported Template end-point

8.3. Synchronous Flow

8.3.1. Apply Template

8.3.1.1. Apply Template URL

Normative URI template of the synchronous template apply end-point per [RFC6570]:

GET

{+urlSyncUX}/v2/domainTemplates/providers/{providerId}/services /{serviceId}/apply{?domain,host,groupId,force,providerName, serviceName,instanceId,redirect_uri,properties*}{&sig,key}

This is the URL, where the user agent (typically web browser) is sent to apply a template to a dns zone the user controls. It is redirected to or linked from the Service Provider to start the synchronous Domain Connect Protocol.

8.3.1.2. Parameters/properties

Property	Request Parameter	Description
URL Sync UX	urlSyncUX	(REQUIRED) Value of urlSyncUX property from the settings endpoint.
Service Provider Id	providerId	(REQUIRED) identifier of the Service Provider of the template to be applied
Service Id	serviceId	(REQUIRED) identifier of the template to be applied
Domain	domain	(REQUIRED) The domain name being configured. This is the Zone Apex (the registered domain or delegated zone).
Host	host	(OPTIONAL) This is the host name of the Sub Domain. If left blank, the template is being applied to the Zone Apex. Otherwise the template is applied to the sub domain of the domain in the same zone.

Redirect URI	redirect_uri	(OPTIONAL) The location to direct the client browser to upon successful authorization, or upon error. If omitted the DNS Provider SHOULD close the browser window upon completion. It MUST be scoped to the syncRedirectDomain authority from the template, or the request MUST be signed.
State	state	(OPTIONAL) A random and unique string passed along to prevent CSRF, or to pass back state. It MUST be returned as a query parameter when redirecting to the redirect_uri described above.
Name/Value Pairs	properties	(REQUIRED) Any key that will be used as a replacement for the "% surrounded" variables in the 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 will be used when applying the template. The client MUST ignore any unknown parameters, not referenced in the template.
Provider Name	providerName	(OPTIONAL) This parameter allows for the caller to provide additional text for display with the template providerName. This text SHOULD be used to augment the providerName value from the template, not replace it. This parameter is only allowed when the "sharedProviderName" attribute is set in the template. Note: this used to be controlled by the "shared" attribute in the template, which has been deprecated.
Service Name	serviceName	(OPTIONAL) This parameter allows for the caller to provide additional text for display with

		the template serviceName. It SHOULD be used to augment the serviceName value from the template, not replace it. This parameter is only allowed when the "sharedServiceName" attribute is set in the template.
Group Id	groupId	(OPTIONAL) This parameter specifies the groups from the template to apply. If no group is specified, all groups are applied. Multiple groups MAY be specified in a comma delimited format.
Signature	sig	(OPTIONAL) A signature of the query string. See Security Considerations section below.
Key	key	(OPTIONAL) A value containing the host in DNS where the public key for the signature can be obtained. The domain for this host is in the template in syncPubKeyDomain. See Security Considerations section below.

Table 5: URI template parameters of the apply call in the sync flow

An example query string:

GET

https://web-connect.dnsprovider.example/v2/domainTemplates/providers/exampleservice.example/services/template1/apply?domain=example.com&IP=192.168.42.42&RANDOMTEXT=shm%3A1542108821%3AHello

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 (exampleservice.example) and the service template owned by them (template1). In this example, there are two variables in this template, "IP" and "RANDOMTEXT". These variables are passed as name/value pairs.

8.3.1.3. Browser handling considerations

The synchronous template apply URL can be called in one of two ways.

8.3.1.3.1. New Browser Window

The first is through a new browser tab or in a popup browser window. The DNS Provider authenticates the user if necessary, verifies domain ownership, and asks for confirmation before application of the template. After application of the template, the DNS Provider should automatically close the browser tab or window.

Please note that in this case the only way Service Provider would know if the user has completed the setup or cancelled the operation is if the user closes the browser window. The Service Provider needs to verify via DNS what actually happened (See: Section 8.5).

8.3.1.3.2. Same Browser Window

The second is in the current browser tab/window. As above the DNS Provider authenticates the user in if necessary, verifies the user control of the DNS Zone for the domain, and asks for confirmation before application of the template. After application of the template (or cancellation by the user), the DNS Provider must redirect the browser to a return URL (redirect_uri query parameter of the Apply Template URL).

Several parameters must be appended to the end of this redirect_uri.

* State

  If a state parameter is passed in on the query string, this must be passed back as state= on the redirect_uri.

* Error

  If authorization could not be obtained or an error has occurred, the parameter error= must be appended. For consistency with the asynchronous OAuth flows the valid values for the error parameter will be as specified in OAuth 2.0 [RFC6749] (4.1.2.1. Error Response - "error" parameter). Valid values are: invalid_request, unauthorized_client, access_denied, unsupported_response_type, invalid_scope, server_error, and temporarily_unavailable.

* Error Description

  When an error occurs, an OPTIONAL error description containing a developer focused error description may be returned.

Under normal operation the access_denied error can be returned for a number of reasons. For example, the user may not have access to the account that owns the domain. Even if they do and successfully sign-in, the account or the domain may be suspended.

It is unlikely that the DNS Provider would want to leak this information to the Service Provider, and as such the description may be vague.

There is one piece of information that may be interesting to communicate to the Service Provider. This is when the end user decided to cancel the operation. If the DNS Provider wishes to communicate this to the Service Provider, when the error=access_denied the error_description may contain the prefix "user_cancel". Again, this is left to the discretion of the DNS Provider.

To prevent an open redirect, unless the request is digitally signed the redirect_uri must be within the domains specified in the template in syncRedirectDomain.

Please note that even though in this case Service Provider is informed whether the process finished without error, by receiving request to redirect_uri without error parameter, the Service Provider still SHOULD verify via DNS to be sure that the changes have been applied (See: Section 8.5).

8.3.2. Security Considerations

8.3.2.1. Risk of phishing with open template parameters

By applying a template with parameters there is a security consideration that must be taken into account.

Consider the template above where the IP address of the A record is passed in through a variable. A bad actor could generate a URL with a malicious IP and phish users by sending out emails asking them to "re-configure" their service. If an end user is convinced to click on this link, they would land on the DNS Provider site to confirm the change. To the user, this would appear to be a valid request to configure the domain. Yet the IP would be hijacking the service.

Not all templates have this problem. But when they do, there are several options.

8.3.2.2. Disable Synchronous

One option is to disable the synchronous flow and use asynchronous OAuth. This can be controlled with the syncBlock value from the template. However, as will be seen below OAuth has a higher implementation burden and requires onboarding between each Service and DNS Provider.

8.3.2.3. Digitally Sign Requests

Another option is to digitally sign the query string. A signature is appended as an additional query string parameter, properly URL encoded and of the form:

sig=V2te9zWMU7G3plxBTsmYSJTvn2vzMvNwAjWQ%2BwTe91DxuJhdVf4cVc4vZBYfEYV
7u5d7PzTO7se7OrkhyiB7TpoJJW1yB5qHR7HKM5SZldUsdtg5%2B1SzEtIX0Uq8b2mCmQ
F%2FuJGXpqCyFrEajvpTM7fFKPk1kuctmtkjV7%2BATcvNPLWY7KyE4%2Bqc8jpfN61cP
5l8iA4krAa3%2BfTro5cmWR8YUJ5yrnRs6KT4b5D71HFvOUk0sGEUddUUlsyRQKRHUFN6
HjEya50YDHfZJlYHkHlK0xX6Yqeii9QZ2I35U9eJbSvZGQko5beqviWFXdsVDbvd3DYcb
SHgJq9%2FXoMTTw%3D%3D

The Service Provider generates this signature using a private key. As indicated, this signature is generated from the query string properly URL encoded.

The Service Provider MUST publish their public key and place it in a DNS TXT record in a domain specified in the template in syncPubKeyDomain at a host of their choice. The TXT record MUST be published as a comma-separated list of key-value properties.

Property	Key	Description
Fragment Index	p	(REQUIRED) The index value of the public key fragment
Fragment Payload	d	(REQUIRED) The payload of public key fragment
Signing Algorithm	a	(OPTIONAL) The parameter identifies the algorithm intended for use with the key. The values used SHALL be registered in the IANA "JSON Web Signature and Encryption Algorithms" registry established by [RFC7518]. If omitted it MUST be assumed to be RS256. The support of RS256 is MANDATORY for both DNS Providers and Service Providers.
Public Key Format	t	(OPTIONAL) The format of the public key. If omitted MUST be assumed to be x509.

Table 6: Properties of the public key TXT record

To allow for key rotation or usage of multiple keys, the host name of the TXT record MUST be appended as another query parameter on the query string of the form:

key=_dcpubkeyv1

This example indicates that the public key can be found by doing a DNS query for a TXT record called _dcpubkeyv1 in the domain specified in the syncPubKeyDomain from the template.

To account for DNS Servers with limits to the size of a TXT record, the public key MAY be split into multiple TXT records at the specified host. For example, a public key of:

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA18SgvpmeasN4BHkkv0SBjAzIc
4grYLjiAXRtNiBUiGUDMeTzQrKTsWvy9NuxU1dIHCZy9o1CrKNg5EzLIZLNyMfI6qiXnM
+HMd4byp97zs/3D39Q8iR5poubQcRaGozWx8yQpG0OcVdmEVcTfyR/XSEWC5u16EBNvRn
NAOAvZYUdWqVyQvXsjnxQot8KcK0QP8iHpoL/1dbdRy2opRPQ2FdZpovUgknybq/6FkeD
tW7uCQ6Mvu4QxcUa3+WP9nYHKtgWip/eFxpeb+qLvcLHf1h0JXtxLVdyy6OLk3f2JRYUX
2ZZVDvG3biTpeJz6iRzjGg6MfGxXZHjI8weDjXrJwIDAQAB

may contain several TXT records. The records would be of the form:

p=1,a=RS256,d=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA18SgvpmeasN
4BHkkv0SBjAzIc4grYLjiAXRtNiBUiGUDMeTzQrKTsWvy9NuxU1dIHCZy9o1CrKNg5EzL
IZLNyMfI6qiXnM+HMd4byp97zs/3D39Q8iR5poubQcRaGozWx8yQpG0OcVdmEVcTfy

p=2,a=RS256,d=R/XSEWC5u16EBNvRnNAOAvZYUdWqVyQvXsjnxQot8KcK0QP8iHpoL/1
dbdRy2opRPQ2FdZpovUgknybq/6FkeDtW7uCQ6Mvu4QxcUa3+WP9nYHKtgWip/eFxpeb+
qLvcLHf1h0JXtxLVdyy6OLk3f2JRYUX2ZZVDvG3biTpeJz6iRzjGg6MfGxXZHjI8

p=3,a=RS256,d=weDjXrJwIDAQAB

Here the public key is broken into three records in DNS, and the data also indicates that the signing algorithm is an RSA Signature with SHA-256. The representation as x509 certificate is the default in this case.

The above data was generated for a query string:

a=1&b=2&ip=10.10.10.10&domain=example.net

Signing the query string by the Service Provider is OPTIONAL. Not all Services Provider templates require or are able to provide this level of security. Presence of the syncPubKeyDomain in the template indicates that the template requires signature verification.

DNS Providers MUST reject any request to apply a template with syncPubKeyDomain present and no signing of a query string in place.

The digital signature MUST be generated on the full query string only, excluding the sig and key parameters. This is everything after the ?, except the sig and key values which MUST be appended to the signed query string.

The values of each query string value key/value pair MUST be properly URL Encoded before the signature is generated.

8.3.2.4. Warnings

Some applications aren’t able to use OAuth and/or sign requests.

If the template require variables, and OAuth and signing isn’t available, the flag warnPhishing SHOULD be set to true in the template in order to offer transparency to the DNS Provider about security properties of the template.

When set this indicates to the DNS Provider that they SHOULD display additional warnings to the user to have them verify the link was/is from a reputable source before applying the template.

8.3.3. Shared Templates

Some templates can be called by multiple companies, or be used for different purposes.

For example, most services are sold and provided by the same company. However, some Service Providers have a reseller channel. This allows the service to be provided by the Service Provider, but sold through third parties. It is often this third party reseller that configures DNS.

While each reseller could enable Domain Connect, this is inefficient for the DNS Providers. Enabling a single template that is shared by multiple resellers would be more optimal.

As another example, some templates may be used for different purposes by the same company.

To facilitate these use cases, the ability to pass in additional context for the display of the providerName and serviceName is enabled. This is only allowed when the template enables the capability through the sharedProviderName and/or sharedServiceName flags.

Note: The shared flag used to be used for this purpose, but has been deprecated.

The exact message presented to the user is up to the DNS Provider. However it is recommended that these fields be used to augment the display of the serviceName and providerName from the template, not replace it.

Note: When a Service Provider has a large reseller channel, it is highly recommended that the Service Provider creates an API for their resellers to ease the implementation of Domain Connect. There are elements of convenience in doing this around Domain Discovery and URL Formatting. But this would be required if the template required signatures.

8.4. Asynchronous Flow: OAuth

8.4.1. General information

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 is used by Domain Connect is given here.

Not all DNS Providers will support the asyncronous flow. As such it is recommended that Service Providers relying on an OAuth implementation also implement a synchronous implementation.

8.4.2. OAuth Flow: Setup

Service Providers wishing to use the OAuth flow MUST register as an OAuth client with each DNS Provider. This is typically a manual process, however other solutions like OAuth Dynamic Client Registration [RFC7591] MAY be offered by DNS Provider as well.

To register, the Service Provider would provide (in addition to their template) any configuration necessary for the DNS Providers OAuth implementation. This includes valid URLs and Domains for redirects upon success or errors of OAuth flow, token validity, presence and validity of refresh tokens etc.

Note: The validity of redirects are very important in any OAuth implementation. Most OAuth vulnerabilities are a combination of an open redirect and/or a compromised secret.

The DNS Provider SHOULD give the Service Provider a client id and a secret which will be used when requesting tokens. For simplicity the client id MAY be the same as the providerId, however it is up to the agreement between the parties involved. Any other form of client authentication within OAuth framework MAY be agreed between the parties.

8.4.3. OAuth Flow: Getting an Authorization Code

Normative URI template of the authorization code end-point per [RFC6570]:

GET

{+urlAsyncUX}/v2/domainTemplates/providers/{providerId}{?domain,host,
client_id,redirect_uri,response_type,scope,providerName,serviceName,
state,properties*}
}

To initiate the OAuth flow the Service Provider first links to the DNS Provider to gain consent.

This endpoint is similar to the synchronous flow described above. The DNS Provider MUST authenticate the user, verify the user has control of the DNS Zone for the domain, and ask the user for permission. Instead of permission to make a change to DNS, the permission is now to allow the Service Provider to make the changes on their behalf. Similarly the DNS Provider MAY warn the user that (the eventual) application of a template might change existing records and/or disrupt existing services attached to the domain.

While the variables for the applied template would be provided later, the values of some variables may be necessary to determine conflicts. As such, any variables impacting conflicting records SHOULD be provided in the consent flow. This primarly includes variables in hosts, and variables in the data portion for certain TXT records.

The protocol allows for the Service Provider to gain consent to apply multiple templates. These templates are specified in the scope parameter. It also allows for the Service Provider to gain consent to apply these templates to the domain or to the domain with multiple sub-domains. These are specified in the domain and host parameter. If conflict detection is implemented by the DNS Provider, they SHOULD account for all permutations, in order to inform the end user of all possible consequences of the authorised change.

The scope parameter is a space separated list (as per the OAuth protocol) of the template serviceIds. The host parameter is an OPTIONAL comma separated list of hosts. A blank entry for the host implies the template can be applied to the Zone Apex For example:

Query String	*Description*
scope=t1%20t2&domain=example.com	Templates "t1" and "t2" can be applied to example.com
scope=t1%20t2&domain=example.com&host=sub1,sub2	Templates "t1" and "t2" can be applied to sub1.example.com or sub2.example.com
scope=t1%20t2&domain=example.com&host=sub1,	Templates "t1" and "t2" can be applied to example.com or sub1.example.com

Table 7: examples of scope and host parameter values in the async flow

Upon successful authorization/verification/consent from the user, the DNS Provider MUST direct the end user’s browser to the redirect URI. The authorization code MUST be appended to this URI as a query parameter of "code=" as per the OAuth specification.

Similar to the synchronous flow, upon error the DNS Provider MAY append an error code as query parameter "error". These errors are also from the OAuth 2.0 [RFC6749] (4.1.2.1. Error Response - "error" parameter). Valid values include: invalid_request, unauthorized_client, access_denied, unsupported_response_type, invalid_scope, server_error, and temporarilly_unavailable. An OPTIONAL error_description suitable for developers may also be returned at the discretion of the DNS Provider. The same considerations as in the synchronous flow apply here.

The state value passed into the call MUST be passed back on the query string as state=.

The following table describes the values of the URI template for the request for the OAuth consent flow that must be included unless otherwise indicated

Property	Key	Description
URL Sync UX	urlAsyncUX	(REQUIRED) Value of urlAsyncUX property from the settings endpoint.
Service Provider Id	providerId	(REQUIRED) identifier of the Service Provider of the template to be applied
Domain	domain	(REQUIRED) The domain name being configured. This is the Zone Apex.
Host	host	(OPTIONAL) An list of comma separated host names upon which the template may be applied. An empty string implies the root.
Client Id	client_id	(REQUIRED) The client id that was provided by the DNS Provider to the Service Provider during registration.
Redirect URI	redirect_uri	(REQUIRED) The location to direct the client’s browser upon successful authorization or upon error. Validation of the redirect_uri MUST be done by the DNS Provider to match the values provided during onboarding.
Response type	response_type	(OPTIONAL) If included it MUST be the string ’code’ to indicate an authorization code is being requested.
Scope	scope	(REQUIRED) The OAuth scope corresponds to the requested templates. This is list of space separated serviceIds.
Provider Name	providerName	(OPTIONAL) This parameter allows for the caller to provide additional text for display with the template providerName. This text SHOULD be used to augment the

		providerName value from the template, not replace it.
Service Name	serviceName	(OPTIONAL) This parameter allows for the caller to provide additional text for display with the template serviceName(s). It SHOULD be used to augment the serviceName value(s) from the template, not replace.
State	state	(OPTIONAL) This is a random, unique string passed along to prevent CSRF or to pass state value back to the caller. If present it MUST be returned as a parameter appended to the redirect_url described above.
Name/ Value Pairs	properties	(OPTIONAL) Any key that will be used as a replacement for the "% surrounded" value(s) in a template required for conflict detection. This includes variables used in hosts and data in certain TXT records.

Table 8: URI template parameters of the authorization end-point in async flow

8.4.4. OAuth Flow: Requesting an Access Token

Normative URI template of the access token end-point per [RFC6570]:

POST

{+urlAPI}/v2/oauth/access_token

Property	Request Parameter	Description
URL API	urlAPI	(REQUIRED) Value of urlAPI property from the settings endpoint.

Table 9: URI template parameters of the access token end-point

Once authorization has been granted, the Service Provider MUST use the Authorization Code provided to request an Access Token. The OAuth specification recommends that the Authorization Code be a short lived token, and a reasonable recommended setting is ten minutes, however the specific setup would depend on specifics of DNS Provider's implementation. 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 typically done via a server to server API call from the Service Provider to the DNS Provider using a POST. When called in this manner a secret is provided along with the Authorization Code.

OAuth does allow for retrieving the access token without a secret. This is typically done when the OAuth client is a client application. When onboarding with the DNS Provider this would need to be enabled.

When the secret is provided (which is the normal case), care must be taken. A malicious user could create a domain that returns a false _'domainconnect' TXT record, and subsequently a JSON call to their own server for the API end point. By doing so, they could then run Domain Connect on their domain and retrieve the secret.

As such the urlAPI used for OAuth by the Service Provider SHOULD be maintained per DNS Provider and not the value retrieved during discovery or other measures have to be implemented to prevent token leakage.

The following table describes the POST parameters that MUST be included in the request for the access token unless otherwise indicated. The parameters SHALL be accepted via the query string or the body of the post. This is again particularly important for the client_secret, as passing secrets via a query string is generally frowned upon given that various systems often log URLs.

The body of the post is application/json encoded.

Property	Key	Description
Authorization Code/Refresh Code	code/ refresh_token	(REQUIRED) The authorization code that was provided in the previous step when the user accepted the authorization request, or the refresh_token for a subsequent access token.
Redirect URI	redirect_uri	(OPTIONAL) This is REQUIRED if a redirect_uri was passed to request the authorization code. When included, it needs to be the same redirect_uri provided in this step.
Grant type	grant_type	(REQUIRED) The type of code in the request. Usually the string 'authorization_code' or 'refresh_token'
Client ID	client_id	(REQUIRED) This is the client id that was provided by the DNS Provider to the Service Provider during registration
Client Secret	client_secret	(REQUIRED) The secret provided to the Service Provider during registration. Typically required unless the rare circumstance with secret-less OAuth.

Table 10: parameters of the token end-point

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

Property	Description
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.

Table 11: properties of the token end-point response

Status	Response	Description
Success	2xx	A response of an http status code of 2xx indicates that the call was successful. The response is the JSON described above.
Errors	4**	All other responses indicate an error.

Table 12: http status codes of the token end-point response

8.4.5. OAuth Flow: Making Requests with Access Tokens

Once the Service Provider has the access token, they can call the DNS Provider's API to make changes to DNS on the domain by applying and (OPTIONALLY) removing authorized templates. These templates can be applied to the Zone Apex or to any Sub Domain that has been authorized.

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
```

While the calls below do not have the same security consideration of passing the secret, it is recommend that the urlAPI be from a stored value vs. the value returned during discovery here as well.

8.4.6. OAuth Flow: Apply Template to Domain.

Normative URI template of the asynchronous apply end-point per [RFC6570]:

POST

{+urlAPI}/v2/domainTemplates/providers/{providerId}/services /{serviceId}/apply{?domain,host,groupId,force,providerName, serviceName,instanceId,properties*}

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

While the providerId is implied in the authorization, this is on the path for consistency with the synchronous flows and other APIs. If not matching what was authorized, an error MUST be returned.

When applying a template to a domain, it is possible that a conflict may exist with previous settings. While it is recommended that conflicts be detected when the user grants consent, because OAuth is asynchronous it is possible that a new conflict was introduced by the user.

While it is up to the DNS Provider to determine what constitutes a conflict (see section on Conflicts below), when one is detected calling this API MUST return an error. This error SHOULD enumerate the conflicting records in a format described below.

Because the user often isn’t present at the time of this error, it is up the Service Provider to determine how to handle this condition. Some providers may decide to notify the user. Others may decide to apply their template anyway using the "force" parameter. This parameter will bypass error checks for conflicts, and after the call the service will be in its desired state.

Calls to apply a template via OAuth require the following parameters posted to the above URL unless otherwise indicated. The DNS Provider MUST accept parameters in query string or body of this post.

The body is application/json encoded.

Property	Key	Description
URL API	urlAPI	(REQUIRED) Value of urlAPI from the settings endpoint.
Service Provider Id	providerId	(REQUIRED) identifier of the Service Provider of the template to be applied
Service Id	serviceId	(REQUIRED) The name or identifier of the template to be applied.
Domain	domain	(REQUIRED) The Zone Apex domain name being configured. It MUST match the domain that was authorized in the token.
Host	host	(OPTIONAL) The host name of the Sub Domain that was authorized in the token. If omitted or left blank, the template is being applied to the Zone Apex.
Name/Value Pairs	*	(REQUIRED) 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 MUST be used when applying the template as per the implementation notes.
Group ID	groupId	(OPTIONAL) Specifies the group of changes in the template to apply. If omitted, all changes are applied. This can also be a comma separated list of groupIds.
Force	force	(OPTIONAL) Specifies that the template SHOULD be applied independently of any conflicts that may exist on the domain. This can be a value of 0 or 1.
Provider Name	providerName	(OPTIONAL) This parameter allows for the caller to provide

		additional context for the providerName that applied the template. It MAY be used by DNS Providers that want to display state regarding which templates have been applied. It is only allowed when the "sharedProviderName" attribute is set in the template being applied.
Service Name	serviceName	(OPTIONAL) This parameter allows for the caller to provide additional context for the serviceName that applied the template. It MAY be used by DNS Providers that want to display state regarding which templates have been applied. It is only allowed when the "sharedProviderName" attribute is set in the template being applied.
InstanceId	instanceId	(OPTIONAL) Only applicable to templates supporting multiple instances (see multiInstance (Section 9.2) template property). Allows for later removal of one template instance by DNS Providers storing this information.

Table 13: URI template parameters of the apply end-point in the async flow

An example call is below. In this example, it is contemplated that there are two variables in this template, "IP" and "RANDOMTEXT" which both require values. These variables are passed as name/value pairs.

POST

https://connect.dnsprovider.example/v2/domainTemplates/providers/exampleservice.example/services/template1/apply?IP=192.0.2.42&RANDOMTEXT=shm%3A1542108821%3AHello&force=1

The API MUST validate the access token, and that the domain belongs to the user and is represented by the token being presented. Any errors with variables, conflicting templates, or problems with the state of the domain are returned; otherwise the template is applied.

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

Status	Response	Description
Success	2xx	Any 200 level code MUST be considered a success. The response MAY be of status 200 with a response body, but also 204 without a body.
Bad Request	400	A response of a 400 indicates that the server cannot process the request because it was malformed or had errors. This response code is intended for programming errors.
Unauthorized	401	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.
Conflict	409	This indicates that the call was good, and the caller authorized, but the change could not be applied due to a conflicting template. Errors due to conflicts MUST NOT be returned when force is equal to 1.
Error	4xx	Other 4xx error codes SHOULD be returned when something is wrong with the request that makes applying the template problematic; most often something that is wrong with the account and requires attention.

Table 14: http status codes of the apply end-point in the async flow

When a 409 is returned, the body of the response SHOULD contain details of the conflicting records. If present this MUST be JSON containing the error code, a message suitable for developers, and an array of tuples containing the conflicting records type, host, and data element.

As an example:

```
{
    "code": "409",
    "message": "Conflicting records",
    "records": [
        {
            "type": "CNAME",
            "host": "www",
            "data": "@"
        },
        {
            "type": "A",
            "host": "@",
            "data": "random ip"
        }
    ]
}
```

In this example, the Service Provider tried to apply a new hosting template. The domain had an existing service applied for hosting.

8.4.7. OAuth Flow: Revert Template

This call reverts the application of a specific template from a domain.

Implementation of this call is OPTIONAL. If not supported a 501 MUST be returned.

Normative URI template of the asynchronous template revert end-point per [RFC6570]:

POST

{+urlAPI}/v2/domainTemplates/providers/{providerId}/services /{serviceId}/revert{?domain,host,instanceId}

This API allows the removal of a template from a user domain/host using an OAuth request.

The provider and service name in the URL MUST match the values provided during authorization.

This call MUST validate that the template exists and has been applied to the domain by the Service Provider, or an error response with code 410 SHOULD be returned that the call would have no effect.

An example URL might look like:

POST

https://connect.dnsprovider.example/v2/domainTemplates/providers
/exampleservice.example/services/template1/revert?domain=example.com

Allowed parameters:

Property	Key	Description
URL API	urlAPI	(REQUIRED) Value of urlAPI from the settings endpoint.
Service Provider Id	providerId	(REQUIRED) identifier of the Service Provider of the template to be applied
Service Id	serviceId	(REQUIRED) The name or identifier of the template to be applied.
Domain	domain	(REQUIRED) The Zone Apex domain name being configured. It MUST match the domain that was authorized in the token.
Host	host	(OPTIONAL) The host name of the Sub Domain that was authorized in the token. If omitted or left blank, the template is being applied to the Zone Apex.
InstanceId	instanceId	(OPTIONAL) Only applicable to templates supporting multiple instances (see multiInstance (Section 9.2) template property). For DNS Provider storing information about applied templates allows removal of single instance of template. If missing all instances of template MUST be removed.

Table 15: URI template parameters of the revert end-point in the async flow

The DNS Provider MUST be able to accept these on the query string or in the body of the POST with application/json encoding.

Response codes Success, Authorization, and Errors are identical to above with the addition of the 501 code.

8.4.8. OAuth Flow: Revoking access

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

8.5. Verification of Changes

There are circumstances where the Service Provider may wish to verify that the template was successfully applied. Without Domain Donnect, this typically involved the Service Provider querying DNS to see if the changes to DNS had been made.

This same technique works with Domain Connect, and if necessary can be triggered either manually on the Service Provider site or automatically upon page/window activation in the browser when the browser window for the DNS Provider is closed.

When the redirect_uri is used and an error is not present in the URI, the Service Provider can not assume the changes were applied to DNS. While true in most circumstances, users can tamper with or alter the return url in the browser. As such it is recommend that enablement of a service be based on verification of changes to DNS.

9. Domain Connect Objects and Templates

9.1. Template Versioning

If a breaking change is made to a template it is recommended that a new template be created. While on the surface versioning looks appealing, in reality this is rarely needed.

Any changes to the template need to account for existing users with settings in DNS, some applied through Domain Connect and some manual. So when changes are made, they are often backward compatible.

Note that when a template changes, it does need to be on-boarded with the DNS Providers.

The version field (Section 9.2) of the template definition serves the purpose of transparency between the DNS Provider and the Service Provider in case of such changes.

9.2. Template Definition

A template is defined as a standard JSON data structure containing the following data. Field values MUST be defined unless otherwise indicated.

Data Element	Type	Key	Description
Service Provider Id	String	providerId	(REQUIRED) 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 SHOULD be the domain name of the Service Provider (e.g. exampleservice.example).
Service Provider Name	String	providerName	(REQUIRED) The name of the Service Provider suitable for display. This SHOULD be displayed to the user on the DNS Provider consent UX.
Service Id	String	serviceId	(REQUIRED) The name or identifier of the template. This is used in URLs to identify the template. It is also used in the scope parameter for OAuth. It MUST NOT contain space characters, and SHOULD be URL friendly.
Service Name	String	serviceName	(REQUIRED) The name of the service suitable for display to the user. This SHOULD be displayed to the user on the DNS Provider consent UX.

Version	Integer	version	(OPTIONAL) If present this represents a version of the template and SHOULD be changed with each update of the template content. This opaque value is mainly informational to improve communication and transparency between providers.
Logo	String	logoUrl	(OPTIONAL) A graphical logo representing the Service Provider and/or Service for use in any web-based flow. If present this MAY be displayed to the user on the DNS Provider consent UX.
Description	Text	description	(OPTIONAL) A textual description of what this template attempts to do. This is meant to assist developers and MUST NOT be displayed to the user.
Variable Description	Text	variableDescription	(OPTIONAL) A textual description of what the variables are. This is meant to assist developers and MUST NOT be displayed to the user.
Synchronous Block	Boolean	syncBlock	(OPTIONAL) Indicates that the synchronous protocol MUST be disabled for this template. The default for this is false.
Shared	Boolean	shared	(OPTIONAL) This flag has been deprecated. It used to indicate that

			the template allowed a dynamic providerName on the query string. It is replaced with the sharedProviderName flag in v2.2 of the spec.
Shared Provider Name	Boolean	sharedProviderName	(OPTIONAL) This flag indicates that the template allows the caller to pass in additional information for the providerName. This information SHOULD augment the display of the providerName from the template. The default for this is false. For backward compatability with DNS Providers not at V2.2 of the spec it is recommended that the shared flag also be set.
Shared Service Name	Boolean	sharedServiceName	(OPTIONAL) This flag indicates that the template allows the caller to pass in additional information for the serviceName. This information SHOULD augment the display of the serviceName from the template. The default for this is false.
Synchronous Public Key Domain	String	syncPubKeyDomain	(OPTIONAL) When present, indicates that calls to apply a template synchronously MUST be digitally signed. The value indicates the domain name for querying the TXT record from DNS that contains the public key used for signing.

Synchronous Redirect Domains	String	syncRedirectDomain	(OPTIONAL) When present, this is a comma separated list of domain names for which redirects are allowed be sent to after applying a template for the synchronous flow. DNS Provider MUST NOT send redirects to any other domain name not included on this list.
Multiple Instance	Boolean	multiInstance	(OPTIONAL) Defaults to False. When set to True, it indicates that the template MAY be applied multiple times. This only impacts DNS Providers that maintain template state in DNS.
Warn Phishing	Boolean	warnPhishing	(OPTIONAL) When present, this tells the DNS Provider that the template likely contains variables susceptible to phishing attacks and the provider is unable to digitally sign the requests. When set the DNS Provider SHOULD display warnings to the user and be more verbose about the changes applied. The default value for this is false.
Host Required	Boolean	hostRequired	(OPTIONAL) Defaults to false. When present this indicates that the template has been authored to work only when both domain and host are provided. An example where this would be true would be a template where CNAME is

			set on the fully qualified domain name. This is largely informational, as most DNS Providers already enforce such rules on the DNS level anyway.
Template Records	Array of Template Records	records	(REQUIRED) A list of records for the template.

Table 16: properties of the template definition

9.3. Template Record

Each template record is an entry that contains a type and several other values depending on the type.

Many of these values can contain variables, which are expressed as strings surrounded with "%" or special variable "@" (See: Section 10.9). Variables are replaced with values when the template is applied.

It is noted that as a best practice the variable portions SHOULD be constrained to as small as possible a portion of the resulting DNS record.

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 %var%.example.com and a value of 01, 02, or 03 passed in. By placing more fixed data into the template, the template is more secure.

Each record MUST contain the following elements unless otherwise specified.

Data Element	Type	Key	Description
Type	enum	type	(REQUIRED) Describes the type of record in

			DNS, or the operation impacting DNS. Valid values include: A, AAAA, CNAME, MX, TXT, SRV, or SPFM. The DNS Provider MUST support the core set of records A, AAAA, CNAME, MX, TXT, SRV. The DNS Provider SHOULD support SPFM record for high interoperability with existing templates
			All other record types MAY be specified by type name as listed in IANA registry for DNS Resource Record (RR) TYPEs. Unknown record types MAY be specified as per [RFC3597] by the word "TYPE" immediately followed by the decimal RR type number, with no intervening whitespace. Support for other record types is OPTIONAL. For each type, additional fields would be REQUIRED. * A: host, pointsTo, TTL * AAAA: host, pointsTo, TTL * CNAME: host, pointsTo, TTL (host MUST NOT be null or @ unless hostRequired is defined true for the template) * NS: host, pointsTo, TTL (host MUST NOT be null or @ unless hostRequired is

			defined true for the template) * TXT: host, data, TTL, txtConflict-MatchingMode, txtConflict-MatchingPrefix * MX: host, pointsTo, TTL, priority * SRV: name, target, TTL, priority, protocol, service, weight, port * SPFM: host, spfRules * other record types: host, data, TTL
Group Id	String	groupId	(OPTIONAL) This parameter identifies the group the record belongs to when applying changes. This MUST NOT contain variables.
Essential	enum	essential	(OPTIONAL) This parameter indicates how the record is treated during conflict detection with existing templates. If the DNS Provider is not implementing applied template state in DNS this is ignored. Always (default) - record MUST be applied and kept with the template OnApply - record MUST be applied but can be later removed without dropping the whole template
Host	String	host	(REQUIRED) The host

			for A, AAAA, CNAME, NS, TXT, MX and other unspecified record type values. This value is relative to the applied host and domain, unless trailed by a ".". A value of empty or @ indicates the root of the applied host and domain. In other words "[host.]example.com.". This value SHOULD NOT contain variables unless absolutely necessary. This is discussed below.
Name	String	name	The name for the SRV record. This value is relative to the applied host and domain. A value of empty or @ indicates the root of the applied host and domain. This value SHOULD NOT contain variables unless absolutely necessary. This is discussed below.
Points To	String	pointsTo	The pointsTo location for A, AAAA, CNAME, NS and MX records. A value of empty or @ indicates the host and domain name being applied or [host.]example.com
TTL	Int or string repr. of Int	ttl	The time-to-live for the record in DNS. Valid for A, AAAA, CNAME, NS, TXT, MX,

			and SRV records. In order to avoid operational unpredactibility of the template and the challenges outlined below this SHOULD NOT contain variables unless absolutely necessary. If it does, the variable MUST be included as string in the template definition to build a valid JSON and the variable MUST be the only value content. Prefixes, suffixes or multiple variables MUST NOT be used. This value, no matter if variable or constant, is understood as "best effort" by DNS Provider and MAY be limited or adjusted by local policy at runtime or during template onboarding, like applying a certain minimum or maximum value of TTL or an enumeration of TTL values supported by the DNS Provider. The DNS Provider SHOULD NOT reject template application because of invalid value, rather pick the nearest supported value or a default, in order to avoid necessity of per provider adjustment to the application flow. Support of variables

			in this field is OPTIONAL for DNS Provider.
Data	String	data	The data for a TXT record in DNS. A value of empty or @ indicates the host and domain name being applied or [host.]example.com For any unspecified record type this field contains the canonical presentation format of the given record. The representation SHALL follow [RFC3597] as generic or type-specific encoding. This MUST NOT be used for any record type explicitly listed in the Type field with specific data fields.
TXT Conflict Matching Mode	String	txtConflictMatchingMode	Describes how conflicts on the TXT record are detected. Possible values are None, All, or Prefix. The default value is None. See below (Section 10.3).
TXT Conflict Matching Prefix	String	txtConflictMatchingPrefix	The prefix to detect conflicts when txtConflict-MatchingMode is "Prefix". This MUST NOT contain variables. See below (Section 10.3).
Priority	Int or string repr. of Int	priority	The priority for an MX or SRV record. This MAY contain variable but if it does the

			variable MUST be included as string in the template definition to build a valid JSON and the variable MUST be the only content of the value field. Prefixes, suffixes or multiple variables MUST NOT be used. Support of variables in this field is OPTIONAL for DNS Provider.
Weight	Int or string repr. of Int	weight	The weight for the SRV record. This MAY contain variable but if it does the variable MUST be included as string in the template definition to build a valid JSON and the variable MUST be the only content of the value field. Prefixes, suffixes or multiple variables MUST NOT be used. Support of variables in this field is OPTIONAL for DNS Provider.
Port	Int or string repr. of Int	port	The port for the SRV record. This MAY contain variable but if it does the variable MUST be included as string in the template definition to build a valid JSON and the variable MUST be the only content of the value field.

			Prefixes, suffixes or multiple variables MUST NOT be used. Support of variables in this field is OPTIONAL for DNS Provider.
Protocol	String	protocol	The protocol for the SRV record.
Service	String	service	The symbolic name for the SRV record.
Target	String	target	The target for the SRV record.
SPF Rules	String	spfRules	These are desired rules for the SPF TXT record. These rules SHOULD be merged with other SPFM records into final SPF TXT record. See Section 10.10.

Table 17: properties of the template record definition

10. Template Considerations

10.1. Template State in DNS

DNS Providers may choose to maintain state inside records in DNS indicating the templates writing the records.

A DNS Provider that maintains this state may be able to provide an improved experience for users, telling them the services enabled. They also may be able to have more advanced handling of conflicts.

To make the implementation burden reasonable for DNS Providers, Domain Connect does not dictate the approach.

10.2. Disclosure of Changes and Conflicts

It is left to the discretion of the DNS Provider to determine what is disclosed to the user when granting permission and/or applying changes to DNS. This includes disclosing the records being applied and the records that may be overwritten.

For changes being made, one DNS Provider may decide to simply tell the user the name of the service being enabled. Another may decide to display the records being set. And another may progressively display both.

For conflict detection, one DNS Provider may simply overwrite changed records without warning. Another may detect conflicts and warn the user of the records that will change. And another may implement logic to further detect, warn, and remove any of the existing templates that overlap with the new template once applied (this assumes they are a DNS Provider that maintains template state in DNS).

As an example, consider applying a template that sets two records (recordA and recordB) into a zone. Next consider applying a second template that overlaps with the first template (recordB and recordC). If the DNS maintains template state and removes conflicting templates, applying the second template would remove the first template. Application of the second template would conflict with recordB and the entire first template would be removed.

Manual changes made by the user at the DNS Provider may also have appropriate warnings in place to prevent unwanted changes; with overrides being possible and removal of conflicting templates.

For the synchronous flow, this happens while the user is present.

For the asynchronous flow, the consent UX is similar. However, the changes are made later using the API and OAuth. The DNS Provider MAY decide to detect conflicts and return these from the API without applying the change using the proper response code. If the force parameter is set, the changes MUST be applied regardless of conflicts.

It is ultimately left to the DNS Provider to determine the amount of disclosure and/or conflict detection. The only requirement is that after a template is applied the new records MUST be applied in totality.

A reasonable set of recommendations for the UX might consist of:

* The consent UX SHOULD inform the user of the service that will be enabled. If the user want to know the specifics, the DNS Provider could provide a "show details" link to the user. This could display to them the specific records that are being set in DNS.

* If there are conflicts, either at the template or record level, the consent UX SHOULD warn the user about these conflicts. For templates, this would be services that would be disabled. For records, this would be records that would be deleted or overwritten. This could be progressively disclosed.

10.3. Record Types and Conflicts

Conflict detection done by the DNS Provider prior to template application has to take into consideration specifics of each DNS record type. The rules outlined below ensure predictable conflict resolution between DNS Providers. Each rule applies to the records on the very same host, unless specifed otherwise.

* CNAME record conflicts with TXT, MX, AAAA, A and existing CNAME records, and any other records of these types conflict with an existing CNAME record. Note: CNAME records cannot be at the root of the zone.

* NS records conflict with all other records. This includes of the same host, and for any record ending with the NS host. For example, an NS record of foo will conflict with any foo, www.foo, bar.foo, etc. Similarly all other record type conflict with NS records in the same manner.

* MX, SRV records always conflict with records of the same type

* A and AAAA records conflict with any other A and/or AAAA record, to avoid IPv4 and IPv6 pointing to different services.

* TXT records conflict detection is handled looking at txtConflictMatchingMode parameter

  - None: This indicates that the TXT records do not conflict with any other TXT record. This is the default setting, if not specified.

  - All: This indicates that the TXT records conflict with any other TXT record

  - Prefix: This indicates that TXT record conflict with any other TXT containing value starting with txtConflictMatchingPrefix

10.4. Apply, Re-apply, and Multi-Instance

There is an additional consideration for DNS Providers that maintain the state of an applied template when re-applying a template.

To avoid unnecessary conflict warnings to the user, under normal use when re-applying a template such a DNS Provider SHOULD remove the previously applied template on the same host.

This may not be desireable for all templates, as a limited set of templates are designed to be applied multiple times. To faciliate this the template can have the flag multiInstance (Section 9.2) set. This tells the DNS Provider that the template is expected to be written multiple times and that a re-apply MUST NOT remove previous instances.

This setting only impacts DNS Providers that maintain applied template state. DNS Providers that do not maintain applied template state can only rely on the normal conflict resolution rules, and this flag has no impact.

10.5. Non-essential records

Typically a template specifies a list of DNS records which are required for the service. There may be cases where some records are only required for a very short period of time, and removing or altering the record later (either by the end user or through application of another template) should not trigger conflict detection.

This can be controlled by the essential (Section 9.3) property of a record in the template.

Again, this setting only impacts DNS Providers that maintain applied template state.

10.6. Template Scope

For DNS Providers that maintain template state, an individual template is scoped to the set of records applied to a fully qualified domain. This includes the Zone Apex and the host (aka Sub Domain) at apply time.

As an example, if a template is applied on domain=example.com&host=sub1 a later application of the template on domain=example.com&host=sub2 must be treated as a distinct template. If a conflict is detected later with the records set into "sub2.example.com", only the records set with this template would be removed.

10.7. Host/Name in Template

Template records contain the host name of the record to set into the zone (called name for SRV records). This value MUST be considered relative to the domain/host when the template is applied, unless followed by a trailing ".".

Consider a template record of type A with a host value of "xyz". When the template is applied to a domain=example.com and an empty host value, the resulting zone after the template is applied will contain an A record of "xyz" (or "xyz.example.com." as absolute domain name in DNS master file notation).

If the same template is applied to a domain=example.com and host=bar, the zone will contain an A record of "xyz.bar" (or "xyz.bar.example.com." as absolute domain name).

A value of @ for host in the template is a placeholder for an empty value. In other words @ would point to "bar.example.com." when the same template is applied to domain=example.com and host=bar.

10.8. PointsTo in Template

Template records of certain types contain the pointsTo value to set in the zone. For record types such as CNAME where this can be a fully qualified domain name.

A value of @ in pointsTo field in the template is a shortcut for the fully qualified domain name of the domain/host being applied.

Consider a template record of type CNAME with a pointsTo value of "@". After a template of domain=example.com and an empty host is applied, the pointsTo value (or corresponding field) in the resulting zone would be "example.com". After a template of domain=example.com with host=bar is applied, the points to value would be "bar.example.com".

Any domain in a pointsTo field in a template MUST be considered fully qualified and not relative.

10.9. Variables

10.9.1. Variable Syntax

Variable expressions are the parameterized parts of a Domain Connect Template. Each expression contains one variable specifier (which can be either a named variable or a special variable "@") that is replaced with a value during template application.

```
variable-expression    =  named-variable / template-apex-var
named-variable         = "%" variable-name "%"
template-apex-var      = "@"
variable-name          =  1*(ALPHA / DIGIT / "-" / "_" )
```

10.9.2. Special and Built-In Variables

There are three Built-In variables:

* %host%: This is the host passed from the query string

* %domain%: This is the domain passed from the query string

* %fqdn%: This is the fully qualified domain name or template application e.g. [host.]domain

For example, with the query string "domain=example.com=", %fqdn% in a template would be "example.com", and with "domain=example.com=sub1", %fqdn% in a template would be "sub1.example.com".

The @ variable has special meaning, and can be used in the host/name field or in the pointsTo/data field in isolation.

For the host/name field it is a shortcut for the value "%fqdn%.". The trailing dot here is equal to the DNS master file notation [RFC1035], which indicates the value is absolute. Without the trailing ".", the value in this field is relative to the [host.]example.com value.

For the pointsTo/data field it is a shortcut for for the "%fqdn%". The pointsTo and data files are always absolute for these fields.

10.9.3. Variable substitution

10.9.3.1. Input Values

Input values for variable substitution MUST be treated as strings. While the underlying data source (e.g., query string, JSON string) might represent values in different data types, the Domain Connect protocol mandates that these values are interpreted and substituted as strings within the templates.

If the data source requires encoding of certain characters (e.g., special characters, spaces), the DNS Provider implementation MUST handle decoding of the value before variable substitution. The resulting substituted value MUST reflect the exact original input value string.

10.9.3.2. Processing

When a template is applied, the variables in the template are replaced with the values passed as input.

Variables are only allowed in template fields of type string, therefore the input field values from the template MUST be decoded from JSON string encoding before variable substitution.

Variables are replaced in the template fields in the order they are found. If a variable is not found in the input, the processing MUST fail. After a variable is replaced, only the remaining string is used for further variable substitution.

The result of the processing MAY still contain variable expressions coming from Input Values of variable substitution. The processing MUST NOT fail in this case, and the variable expressions MUST be left as is.

10.9.4. Variables and Host/Name in Template

While templates do allow for variables in a host or name field values, these SHOULD be used very sparingly.

As an example, consider setting up hosting for a site. But instead of applying the template to a domain/host, the name of the host is placed as a variable in the template.

Such a template might contain an A record of the form:

```
{
    "type": "A",
    "host": "%var%",
    "pointsTo": "192.0.2.2",
    "ttl": 1800
}
```

This template could be applied on a domain like example.com with the var set to "sub", "sub1", "sub2", etc.

Application of this template would be at the domain level for "example.com". This causes problems for application/re-application of the template, conflict detection, and template removal.

Since this template would be applied to the domain only, DNS Providers that maintain template state would remove previous instances of the template before re-application. This means applying this template with var=sub would result in the A record for sub.example.com to be set to the value 192.0.2.2. Later, applying the template on "example.com" with the var=sub2 should remove the old template before setting the new one. sub.example.com would be removed, and sub2.example.com would be set to the value 192.0.2.2.

Furthermore, determining conflicts would be impossible when the user is granting consent for asynchronous operations (OAuth). This is because the host would be indeterminate.

To solve this problem, templates MUST be considered as scoped to a domain and a host value. For synchronous operations, the host value is specified in the url. For asynchronous operations, permissions are granted for specific host values, whose value is later specified when applying the template.

Some templates might want to utilize CNAME or TXT records with host values containing some form of user identification for validation of domain ownership, and these are often passed in variables. In those cases usage of variables in host field value can be applied without risk of issues mentioned above. In all other cases it is RECOMMENDED not to define variables in the host field.

10.9.5. Variables and Security

As discussed, with variables consideration is necessary to prevent certain styles of phishing attacks.

The more static the value in the template record, the more secure the template. When static values are not possible, a carefully crafted link could hijack DNS settings.

Mitigations to this are discussed above.

10.9.6. Variable Examples

Example template:

```
[{
    "type": "CNAME",
    "host": "www",
    "pointsTo": "@",
    "ttl": 1800
},
{
    "type": "A",
    "host": "@",
    "pointsTo": "192.0.2.1",
    "ttl": 1800
}]
```

Template applied with domain=example.com and host parameter missing or empty:

```
www 1800 IN CNAME example.com.
@   1800 IN A 192.0.2.1
```

alternatively

```
www.example.com.    1800 IN CNAME example.com.
example.com.        1800 IN A 192.0.2.1
```

Template applied with domain=example.com and host=bar:

```
www.bar 1800 IN CNAME bar.example.com.
bar     1800 IN A 192.0.2.1
```

alternatively

```
www.bar.example.com.    1800 IN CNAME bar.example.com.
bar.example.com.        1800 IN A 192.0.2.1
```

10.10. SPF TXT Record

10.10.1. What is SPF?

SPF stands for Sender Policy Framework specified in [RFC7208]. It is a record that specifies a list of authorized host names and/or IP addresses from which mail can originate from for a given domain name.

It manifests itself as a TXT record. The format of which starts with v=spf1 followed by a list of "rules" of what to include/exclude. If a rule passes, the mail is allowed. If it fails, it moves to the next rule. Typical record might appear as:

v=spf1 include:policy.exampleprovider.example -all

This is an SPF record with two rules. The first rule indicates that the rules for SPF record _policy.exampleprovider.example be included in this record. The second rule is a catch all (all). The default modifier for a rule is pass (+). Other modifiers are hard failure (-), soft failure (~) and neutral (?).

Note: A failure in SPF doesn’t mean delivery won’t happen, however depending on the policies of the receiving system, messages classified with hard failure or soft failure may not be delivered or marked as spam.

The use of "all" at the end is pretty common, although some providers mark it as ~ (soft fail) or ? (neutral). The reality is that a good SPF record is tuned based on what services are attached to a domain. Not just one individual service.

10.10.2. Multiple Services

If only one email sending service were active, the SPF record recommended by the provider is sufficient. But mail from a domain can often come from several different services.

A very typical use case might be end user mail and an email newsletter service. Let’s look at the SPF records recommended for individual services.

Mailer1: v=spf1 include:spf.mailer1.example -all Newsletter1: v=spf1 include:_spf.newsletter.example ~all

All of these examples use the include syntax. This is fairly common. The use of all at the end is common, although is often inconsistent with the modifier.

If a user installed Mailer1 and Newsletter1, their combined SPF record ought to be something like:

v=spf1 include:spf.mailer1.example include:_spf.newsletter.example ~all

We combined the two rules, and in this case picked the least restrictive all modifier.

10.10.3. SPF Record Merging

The challenge with SPF records and Domain Connect is that an individual service might recommend an SPF record. If only one service were active, this would be accurate. But with several services together only the DNS Provider is able to determine the valid shape of a SPF TXT record.

One solution to this problem is to merge all related records. At the highest level, this means taking everything between the "v=spf1" and the "all" from each of the records and merging them together, terminating with hard-coded modifier on all at the end. For an SPF record to fulfill it’s purpose of protection against malicious email delivery, Domain Connect advises a fixed modifier "~" advising lower rating of the messages from other sources not specified in SPF. This setup offers a reasonable level of protection of mail delivery, on the other side does not reject the message in case forwarding facility is in place.

```
@ TXT v=spf1 include:spf.mailer1.example include:_spf.newsletter.exam
ple ~all
```

The other would be to write intermediate records, and reference these locally.

```
r1.example.com. TXT v=spf1 include:spf.mailer1.example ~all
r2.example.com. TXT v=spf1 include:_spf.newsletter.example ~all
@ TXT v=spf1 include:r1.example.com include:r2.example.com ~all
```

There are advantages and disadvantages to both approaches. SPF records have a limit of 10 DNS lookups and record length is limited to 255 characters. So depending on the embedded records both approaches might have advantages.

The implementation would be left to the DNS Provider, but to facilitate this SPF records SHOULD NOT be included in templates. Instead, a new pseudo-record type is introduced in the template called SPFM. This has the following attribute:

spfRules Determines the desired rules, basically everything but leading "v=spf1" and trailing all rule - see: SPF Rules (Section 9.3)

When a template is added or removed with an SPFM record in the template, some code would need to take the aggregate value of all SPFM records in all templates applied as well as existing SPF TXT record on the host and recalculate the resulting SPF TXT record. In case several sources specify the same rule with a different policy

DNS Provider SHOULD apply the least restrictive one as a result. soft failure SHOULD be preferred over hard failure, neutral SHOULD be preferred over soft failure.

DNS Provider SHOULD also allow the end user to modify the SPF record after merging.

Due to merging step in between, the resulting SPF TXT records are considered non-essential (see: Section 10.5). That means the user may decide to override the final calculated value or remove the whole SPF record. This action MUST NOT lead to removal of any related templates in conflict detection and template integrity routines if implemented by the DNS Provider.

If the existing TXT record makes the merging operation not possible, the DNS Provider MUST handle this situation the same way as a conflict and either let the end-user resolve it in the UX (both in Synchronous and Asynchronous flow) or return the conflict as an error in the Asynchronous flow unless the force=true parameter is used, effectively removing the existing record.

Service Providers MUST NOT check content of TXT SPF record for an exact match, as it might be strongly influenced by the DNS Provider merging strategy and user actions.

See Appendix A.6.

10.10.4. Alternatives

Some DNS Providers MAY decide not to support the SPFM record. The following alternative solution would allow general interoperability of the templates for those providers: onboard the templates with SPFM record in variable-compatible form using a regular TXT record with content "v=spf1 %spfRules% ~all", using property essential=OnApply set to avoid removal of the whole template by a conflict.

10.11. Public Template Repository

10.11.1. General information

The Public Template Repository is an open accessible location where Service Providers MAY publish their Service Templates in the format specified in this specification. DNS Providers MAY support all of the published templates, just a subset or none of them according to own onboarding policies (see also: Section 11.1).

The template format is intended largely for documentation and communication between the DNS Providers and Service Providers, and there are no codified endpoints for creation or modification of these objects. Instead, Domain Connect references a template by ID.

As such, DNS Providers may or may not use templates in this format in their internal implementations. By defining a standard template format, it is believed it will make it easier for Service Providers to share their configuration across DNS Providers.

10.11.2. Repository Location

The repository of the templates is maintained under https://github.com/Domain-Connect/templates.

10.11.3. File naming requirements

The file names in this repository MUST be all lower case, including the providerId and serviceId. As a result, while the providerId and serviceId can be mixed case, all ‘providerIds and serviceIds in this repository MUST be unique when lower case.

Templates MUST be named according the following pattern: {providerId}.{serviceId}.json

providerId: example.com
serviceId: WebsiteBuilder

Template file name: example.com.websitebuilder.json

10.11.4. Template Integrity

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

Hard-coded host names are the responsibility of the DNS Provider to protect. That is, DNS Providers MUST ensure 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.

11. General considerations

11.1. Onboarding

This specification is an open standard that describes the protocol, messages and formats used to enable Domain Connect between a Service Provider and a DNS Provider.

Any Service Provider is free to define and publish a template. However, the terms and conditions for a DNS Provider onboarding a Service Provider template is beyond the scope of this document. A DNS Provider can be selective in what templates they support, can require a contractual relationship, or even charge a fee for onboarding.

One way a Service Provider can be selective in which DNS Providers they accept is to implement a whitelist of providerIds. A Service Provider who chooses to whitelist MUST use providerId to distinguish between unique DNS Providers. The DNS Provider’s providerId would typically be a domain name.

11.2. Case Sensitivity

All values are case sensitive. This includes variable names, values, parameters and objects returned.

One exception is the domain/host name. This is because a fully qualified domain name is case insensitive.

The values for providerId/serviceId in the template and passed through URIs in the path or query string are case sensitive. Different rules apply to the file naming in the Public Template Repository (Section 10.11.3).

12. Extensions/Exclusions

12.1. General information

Additional record types and/or extensions to records in the template can be implemented on a per DNS Provider basis. However, care should be taken when defining extensions so as to not conflict with other protocols and standards. Certain record names are reserved for use in DNS for protocols like DNSSEC (DNSKEY, RRSIG) [RFC9364] at the registry level.

Defining these OPTIONAL extensions in an open manner as part of this specification is done to provide consistency. The following are the initial OPTIONAL extensions a DNS Provider/Service Provider may support.

12.2. APEXCNAME

Some Service Providers desire the behavior of a CNAME record, but in the apex record. This would allow for an A Record at the root of the domain but dynamically determined at runtime.

The recommended record type for DNS Providers that wish to support this is an APEXCNAME record. Additional fields included with this record would include pointsTo and TTL.

Defining a standard for such functionality in DNS is beyond the scope of this specification. But for DNS Providers that support this functionality, using the same record type name across DNS Providers allows template reuse.

12.3. Redirection

Some Service Providers desire a redirection service associated with the A Record. A typical example is a service that requires a redirect of the domain (e.g. example.com) to the www variant (www.example.com). The www would often contain a CNAME.

Since implementation of a redirection service is typically simple, it is recommended that Service Providers implement redirection on their own. But for DNS Providers that have a redirection service, supporting simple templates with this functionality may be desired.

While technically not a "record" in DNS, when supporting this OPTIONAL functionality it is recommended that this SHOULD be implemented using two new record types.

REDIR301 and REDIR302 would implement 301 and 302 redirects respectively. Associated with this record would be a single field called the "target", containing the target url of the redirect.

Please note, that setting up an HTTP redirect typically involves setting up a webserver and configuring A and AAAA records pointing to this webserver accordingly. The template may or may not explicitly define those records, as the orchestration and the values would we controlled by the DNS provider, not the Service Provider. The DNS Provider SHOULD however consider those records during the conflict resolution in order to give transparency of the changes to the user.

12.4. Nameservers

Several Service Providers have asked for functionality supporting an update to the nameserver records at the registry associated with the domain.

When implementing this functionality a template defined at DNS Provider would need to define a set of NS records, typically a minimum of 2 and a maximum depending on the number supported by the registrar system of the DNS Provider, likely with groupIds to allow for optionality of those additional nameservers. Each NS entry would then contain a data field with desired host names or variables to allow for dynamic setup.

It will be noted that a nameserver update would require that the entity implementing DNS Provider side of the protocol is the Registrar. This is not always the case.

Additional care would have to be taken by the DNS Provider informing the user about the change happening and the impact of the change.

This functionality is again deemed as OPTIONAL and up to the DNS Provider to determine if they will support this.

12.5. DS (DNSSEC)

Requests have been made to allow for updates to the DS record for DNSSEC. This record is required at the registry to enable DNSSEC, but can only be written by the registrar.

For DNS Providers that support this record, the record type would be DS. Values will be keyTag, algorithm, digestType, and digest.

Again it should be noted that a DS update would require that the entity implementing DNS Provider side of the protocol is the registrar, and is again deemed as OPTIONAL and up to the DNS Provider to determine if they will support.

13. IANA Considerations

Per [RFC8552], please add the following entry to the "Underscored and Globally Scoped DNS Node Names" registry:

RR Type	_NODE NAME	Reference
TXT	_domainconnect	This document.

Table 18: IANA Registration for _domainconnect

Implementation Status

This section is to be removed before publishing as an RFC.

DNS Providers

Open Source

* Server library (Python): https://github.com/Domain-Connect/DomainConnectApplyZone

Priopriatery implementations

* ~20 providers, incl. GoDaddy, IONOS, Cloudflare, Squarespace Domains (former Google), Wordpress.com or Plesk

* 35% of the .com zone (May’24)

Service Providers

Open Source

* Example service: https://exampleservice.domainconnect.org/ https://github.com/Domain-Connect/exampleservice

* Client library (Python): https://github.com/Domain-Connect/domainconnect_python

Priopriatery implementations

* 300 templates from over 120 providers, incl. O365, Google Workspace, Apple Cloud+, Weebly, Squarespace... https://github.com/Domain-Connect/Templates

Change History

This section is to be removed before publishing as an RFC.

Change from -01 to -02

* Draft refresh from expire. No content changes.

Change from -00 to -01

* Changed term Root Domain to Zone Apex to align with [RFC8499].

* Removed example provider names from Service Providers and DNS Providers teminology

* Added Use Cases
* Added Trust Model
* Added sequence diagrams for synchronous and asynchronous flows instead of UX mocks
* Reviewed use of normative language
* Cleaned up usage of terminology
* Variable substitution description updated
* All URLs are now normatively defined with URI templates

Change from draft-kowalik-regext-domainconnect-00 to draft-kowalik-domainconnect-00

* Added possibility to specify any DNS record type in a generic manner.
* Added possibility to define variables for numeric fields.
* Added IANA registration for _domainconnect record as per [RFC8552]

Change from draft-carney-regext-domainconnect-03 to draft-kowalik-regext-domainconnect-00

* Version synchronized with 2.2 version rev. 66 of the public Domain Connect specification.

Change from -02 to -03

* Added width/height JSON values returned by DNS Provider Discovery.
* Corrected text of GET method for getting the authorization token.
* Added clarifying text to Group ID description parameter of the apply template POST method. Quite a few minor edits and clarifications that were found during implementation, especially in the Implementation Considerations section.

Change from -01 to -02

* Added new GET method for Service Providers to determine if the DNS Provider supports a specific template. Some other minor edits for clarification.

Change from draft-carney-regext-domainconnect-00 to -01

* Minor edits and clarifications found during implementation.

Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", IETF, STD 13, DOI 10.17487/RFC1035, BCP 13, RFC 1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", IETF, DOI 10.17487/RFC8174, BCP 14, RFC 8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC7208] Kitterman, S., "Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1", IETF, DOI 10.17487/RFC7208, RFC 7208, April 2014, <https://www.rfc-editor.org/info/rfc7208>.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", IETF, DOI 10.17487/RFC6749, RFC 6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC3597] Gustafsson, A., "Handling of Unknown DNS Resource Record (RR) Types", IETF, DOI 10.17487/RFC3597, RFC 3597, September 2003, <https://www.rfc-editor.org/info/rfc3597>.

[RFC8552] Crocker, D., "Scoped Interpretation of DNS Resource Records through "Underscored" Naming of Attribute Leaves", IETF, DOI 10.17487/RFC8552, BCP 222, RFC 8552, March 2019, <https://www.rfc-editor.org/info/rfc8552>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", IETF, STD 66, DOI 10.17487/RFC3986, BCP 66, RFC 3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", IETF, DOI 10.17487/RFC7518, RFC 7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", IETF, DOI 10.17487/RFC6570, RFC 6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[RFC5234] Overell, P. and D. Crocker, "Augmented BNF for Syntax Specifications: ABNF", IETF, STD 68, DOI 10.17487/RFC5234, BCP 68, RFC 5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

Informative References

[RFC9364] Hoffman, P., "DNS Security Extensions (DNSSEC)", IETF, DOI 10.17487/RFC9364, BCP 237, RFC 9364, February 2023, <https://www.rfc-editor.org/info/rfc9364>.

[RFC8499] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", IETF, DOI 10.17487/RFC8499, RFC 8499, January 2019, <https://www.rfc-editor.org/info/rfc8499>.

[RFC7591] Jones, M., Bradley, J., Machulak, M., Hunt, P., and J. Richer, "OAuth 2.0 Dynamic Client Registration Protocol", IETF, DOI 10.17487/RFC7591, RFC 7591, July 2015, <https://www.rfc-editor.org/info/rfc7591>.

Appendix A. Examples

A.1. Example Template

```
{
    "providerId": "example.com",
    "providerName": "Example Web Hosting",
    "serviceId": "hosting",
    "serviceName": "Wordpress by example.com",
    "version": 1,
    "logoUrl": "https://www.example.com/images/billthecat.jpg",
    "description": "This connects your domain to our web hosting",
    "records": [
        {
            "type": "A",
            "groupId": "service",
            "host": "www",
            "pointsTo": "%var1%",
            "ttl": 600
        },
        {
            "type": "A",
            "groupId": "service",
            "host": "m",
            "pointsTo": "%var2%",
            "ttl": 600
        },
        {
            "type": "CNAME",
            "groupId": "service",
            "host": "webmail",
            "pointsTo": "%var3%",
            "ttl": 600
        },
        {
            "type": "TXT",
            "groupId": "verification",
            "host": "example",
            "ttl": 600,
            "data": "%var4%"
        }
    ]
}
```

A.2. 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": "192.0.2.1",
    "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 "192.0.2.1"

A.3. 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": "198.51.100.%srv%",
    "ttl": 600
}]
```

A query string with a key/value pair of

srv=2

would cause the application of this template to a domain to set the host name for the apex A record to the IP address "198.51.100.2" with a TTL of 600

A.4. Example Records: Unspecified record type CAA

This example shows how to include a set of unspecified record types on an example of CAA records:

```
[
    {
        "type": "CAA",
        "host": "@",
        "data": "0 issue \"ca1.example.net\"",
        "ttl": 1800
    },
    {
        "type": "CAA",
        "host": "@",
        "data": "0 issuewild \"ca2.example.\"",
        "ttl": 1800
    }
]
```

This would have no variable substitution and the application of this template to a domain would add 2 CAA records.

A.5. Example: DNS Zone merging

Consider a DNS Zone before a template application:

```
$ORIGIN example.com.

@ 3600 IN SOA ns11.example.net. support.example.net. 2017050817 7200
1800 1209600 3600
@ 3600 IN NS ns11.example.net.
@ 3600 IN NS ns12.example.net.
@ 3600 IN A 192.0.2.1
@ 3600 IN A 192.0.2.2
@ 3600 IN AAAA 2001:db8:1234:0000:0000:0000:0000:0000
@ 3600 IN AAAA 2001:db8:1234:0000:0000:0000:0000:0001
@ 3600 IN MX 10 mx1.example.net.
@ 3600 IN MX 10 mx2.example.net.
@ 3600 IN TXT "v=spf1 a include:spf.example.org ~all"
www 3600 IN CNAME other.host.example.
```

Now application of the following template:

```
[
    {
        "type":"A",
        "host":"@",
        "pointsTo":"203.0.113.2",
        "ttl":"1800"
    },
    {
        "type":"A",
        "host":"www",
        "pointsTo":"203.0.113.2",
        "ttl":"1800"
    },
    {
        "type":"SPFM",
        "host":"@",
        "spfRules":"a include:spf.hoster.example"
    }
]
```

The following DNS Zone would be generated after the template is applied:

```
$ORIGIN example.com.

@ 3600 IN SOA ns11.example.net. support.example.net. 2017050920 7200
1800 1209600 3600
@ 3600 IN NS ns11.example.net.
@ 3600 IN NS ns12.example.net.
@ 1800 IN A 203.0.113.2
@ 3600 IN MX 10 mx1.example.net.
@ 3600 IN MX 10 mx2.example.net.
@ 1800 IN TXT "v=spf1 a include:spf.example.org include:spf.hoster.ex
ample ~all"
www 1800 IN A 203.0.113.2
```

A.6. Example: SPF Record Merging

Consider a DNS Zone before a template application:

```
$ORIGIN example.com.

@ 3600 IN SOA ns11.example.net. support.example.net. 2017050817 7200
1800 1209600 3600
@ 3600 IN NS ns11.example.net.
@ 3600 IN NS ns12.example.net.
```

Now application of the following template of Mail service:

```
[
    {
        "type":"MX",
        "host":"@",
        "priority": "10",
        "pointsTo":"mx1.example.net",
        "ttl":"1800"
    },
    {
        "type":"MX",
        "host":"www",
        "priority": "10",
        "pointsTo":"mx2.example.net",
        "ttl":"1800"
    },
    {
        "type":"SPFM",
        "host":"@",
        "spfRules":"a include:spf.example.net"
    }
]
```

Expected result in the DNS Zone

```
$ORIGIN example.com.

@ 3600 IN SOA ns11.example.net. support.example.net. 2017050817 7200
1800 1209600 3600
@ 3600 IN NS ns11.example.net.
@ 3600 IN NS ns12.example.net.
@ 3600 IN MX 10 mx1.example.net.
@ 3600 IN MX 10 mx2.example.net.
@ 3600 IN TXT "v=spf1 a include:spf.example.net ~all"
```

In the next step application of the following template of Newsletter service:

```
[
    {
        "type":"SPFM",
        "host":"@",
        "spfRules":"include:_spf.newsletter.example"
    }
]
```

Expected result in the DNS Zone

```
$ORIGIN example.com.

@ 3600 IN SOA ns11.example.net. support.example.net. 2017050817 7200
1800 1209600 3600
@ 3600 IN NS ns11.example.net.
@ 3600 IN NS ns12.example.net.
@ 3600 IN MX 10 mx1.example.net.
@ 3600 IN MX 10 mx2.example.net.
@ 3600 IN TXT "v=spf1 a include:spf.example.net include:_spf.newslett
er.
example ~all"
```

Authors' Addresses

P Kowalik
DENIC eG
Theodor-Stern-Kai 1
Frankfurt am Main
Germany
Email: pawel.kowalik@denic.de
URI: https://denic.de

A Blinn
Email: arnold@arnoldblinn.com

J Kolker
GoDaddy Inc.
14455 N. Hayden Rd. #219
Scottsdale,
United States of America
Email: jkolker@godaddy.com
URI: https://www.godaddy.com

S Kerola
Cloudflare, Inc.
101 Townsend St
San Francisco,
United States of America
Email: kerolasa@cloudflare.com
URI: https://cloudflare.com