GWD-R.236
NSI-WG
nsi-wg@ogf.org

John MacAuley, ESnet
Guy Roberts, GÉANT
1 June 2019

Network Service Interface Document Distribution Service

Status of This Document

Grid Working Document - Recommendation (GWD-R)

Copyright Notice

Abstract

This document describes the Network Service Interface (NSI) Document Distribution Service (DDS) version 1.0; it is an Application Programing interface (AP) that supports the distribution of meta-data documents throughout an interconnected network of NSI Network Service Agents (NSA) in the Service Plane. The DDS is a REST based API that supports the dynamic distribution of data within the NSI Service Plane by providing a flooding based protocol for exchange of documents published by an NSA about itself and its Networks. By abstracting the DDS for the exchange of meta-data from the meta-data itself, a more generic service is provided which meets the requirements for distribution of NSA Description documents, NSI Topology documents, and NSI Service Definition documents. This document should be read in conjunction with GFD.213, Network Services Framework v2.0 [GFD.213].

Notational Conventions

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” are to be interpreted as described in [RFC 2119]. Words defined in the glossary are capitalized (e.g. Connection). NSI protocol messages and their attributes are written in camel case and italics (e.g. *reserveConfirmed*)

Contents

1 Introduction

1.1 Context

Within the Network Service Framework (NSF) [GFD.213], the Network Service Agent (NSA) is an entity that offers network services. Peer NSA entities communicate using the Network Service Interface (NSI) protocols, a suite of individual protocols that provide the infrastructure needed to offer network services.

These network services need to disseminate meta-data documents which clients require to properly utilize the offered service. One such document is the NSA Description Document [OGF NSI-ND], which is a meta-data schema designed to enable self-description of all NSI services and associated protocol interfaces offered by these NSA. Other information relating to the NSA itself, such as software versions, administrative contacts, location, peerings, and managed networks are also defined as part of the meta-data profile.

Another meta-data document is the NSI topology document. This document provides a description of the resources in the Service Plane based on the NML methodology. [OGF NML] Appendix I: Topology distribution requirements, shows the set of objectives that motivated the DDS. The NSI signaling and pathfinding document [OGF NSI-NSIPF] explains the message flow in NSI and the way in which NSI topology is used for pathfinding.

This type of dynamic data-discovery mechanism is a key element of large-scale distributed systems. By making the NSI protocol and its agents more self-descriptive, new documents, features, protocols, or protocol versions can be added to agents within the Service Plane and then be discovered by peer agents through this meta-data service. As new features come on line, agents supporting the capabilities can discover compatible peer agents, and then negotiate the use of these new features, while older versions of agents within the Service Plane remain unaffected. Similarly, newer versions of agents can still negotiate features and communicate with older agent versions using mutually supported versions of the protocol as described in the discovered meta-data.

The NSI Document Distribution Service is part of the NSF suite of protocols, and is a peer-to-peer flooding protocol for exchange and distribution of many different types of data documents between NSA within the interconnected network or ‘Global Document Space’. It supports both polling and subscription based notification mechanisms for exchange of documents. For the purpose of this recommendation, a *DDS requester* is any application or Network Service Agent (NSA) that is participating as a client in the document distribution protocol (client role). A *DDS provider* is any Network Service Agent (NSA) that is participating in the service as a server for the document space (server role). NSA can participate in both the requester and provider roles of the document distribution service. A DDS requester/provider could also be deployed independent of a Connection Service NSA if so desired.

This recommendation forms a normative of the NSI protocol suite. Where a section of this document is normative, this will be indicated after the section heading.

1.2 Document structure

This document sets out an REST based API for NSI document distribution. Section 2 sets the DDS in the context of the NSI Service Framework. Section 3 then introduces the DDS push and pull methods and explains the message workflow. Section 4 defines the meta-data that is

attached to each NSI documents. Section 5 describes how to use the time-to-live attribute. Section 6 describes how to use subscription mode. Section 7 sets out a formal definition of the DDS API. Section 9 describes the NSI bootstrap procedure and section 9 describes the peer flooding and version sequencing. The full REST profile is set out in section 10.

1.3 Global Document Space

In this document the term 'Global Document Space' (GDS) is defined to be a collection of all documents published within the document space of each provider participating in a DDS deployment. The DDS supports both a push/pull model to distribute/retrieve documents with the GDS. The push model propagates all documents published locally within a provider to all other subscribed providers participating in the GDS. This allows all participating providers to eventually receive a consistent version of all documents within the GDS.

2 DDS in the NSI Service Framework

A basic overview of the functional components of the NSF architecture is described here to provide context to the reader. This section is informational only. Addition detail can be found in [GFD.213].

An NSA is said to be an NSI requester if the NSA is capable of issuing service requests, while it is an NSI provider if it can receive service requests. An NSA may act as both a requester and a provider. The NSF defines three distinct roles for an NSA within the architecture:

- uRA: The ultimate Requester Agent is an NSA that originates but does not respond to service requests. The uRA could, for example, exist in a middleware application.
- uPA: The ultimate Provider Agent is an NSA that services requests by coordinating with the local Network Resource Manager (NRM) to manage network resources. The uPA responds to service requests, but never initiates them.
- AG: The Aggregator Agent is an NSA that has no physical network resources, but can orchestrate end-to-end network services on behalf of a user by utilizing the connection services exposed by an associated uPA or one or more child NSA. By definition the AG is both a requester and a provider NSA.

An AG participating in the NSI Connection Service [OGF NSI-CS] requires access to a number of documents distributed by NSA through the NSI Document Distribution Service to perform basic functions such as:

- Bootstrapping communications with peer NSAs (uRA, uPA, and other AG) using the NSA Description Document [OGF NSI-ND].
- Syntactic Processing and validating parameters parsed using NSI Service Definition Documents [OGF NSI-SD].
- Performing intelligent path finding for a requested connection service using NSI Topology Documents [OGF NSI-TS]. See the Appendix in section 19 for details of the NSI topology distribution requirements.

An ultimate Provider NSA participating in the NSI Connection Service does not require access to documents, but is required to distribute the following documents through the NSI Document Distribution Service:

- An NSA Description Document describing itself in detail, including supported interfaces, features, and networks.
- NSI Service Definition Documents for all services being offered by the local Network managed by the associated NRM.
- NSI Topology Documents of all advertised topology for the local Network managed by the associated NRM.

An ultimate Requester NSA participating in the NSI Connection Service does not produce any documents, however, it can optionally use the following documents from the NSI Document Distribution Service:

- The NSA Description Document from peer provider NSA to discover identity, supported interfaces, features, and networks.
- The NSI Service Definition Documents to determine available service types being offered within the Network.
- The NSI Topology Documents if discovery of network ports or intelligent path finding is implemented by the uRA.

3 Messages and workflow

This section introduces the concepts of the DDS methods and explains the workflow. This section is normative.

The DDS supports both a *getDocuments()*and *addSubscription*() messges. The get message allows a document to be retrieved (pull model). The subscription message allows a DDS requester to register to receive document updates (push model).

3.1 Push and pull methods

A DDS requester utilizes the provider's Document Distribution Service API to query documents stored within the Document Space (DS).

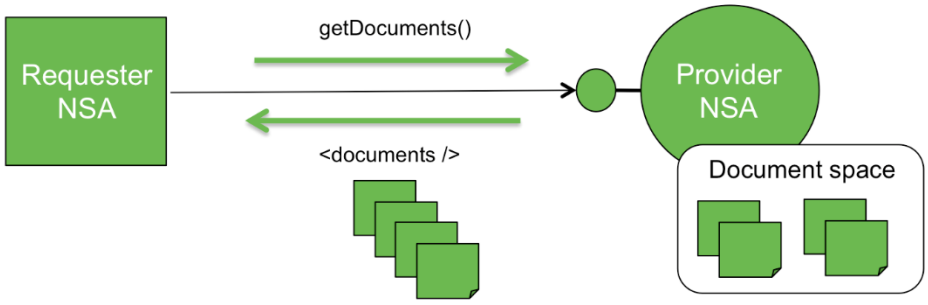

Figure 1 – Simple document get operation.

Figure 1 shows the simple *getDocuments()* operation that is invoked by the DDS requester on the provider NSA to retrieve a set of documents from the document space. These simple document operations follow the standard request/response model.

The DDS requester can also subscribe to document discovery and documents updates within the document space. There is also a Document Distribution Service API to publish, update, and delete documents to/from a local provider.

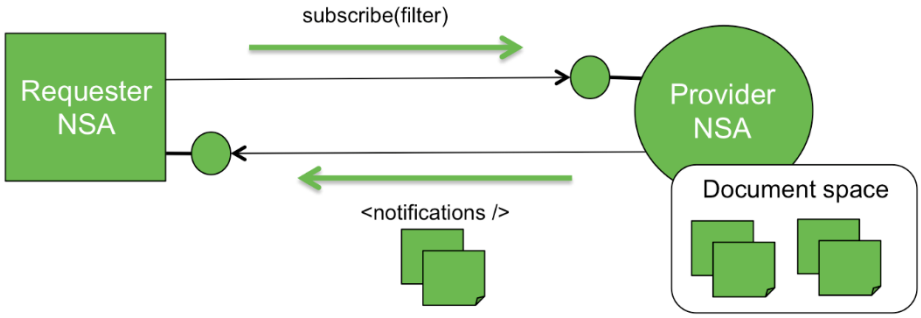

Figure 2 – Document change notification.

Figure 2 illustrates the interaction of the asynchronous publish/subscribe model supported by the document distribution service's notification interface. In this example, the DDS requester requests a subscription supplying a filter to identify the documents of interest. In this subscription request the DDS requester also supplies a callback protocol endpoint that will receive the notifications delivered from the provider NSA. When there is a document event matching the subscription filter, the provider NSA will deliver the document to the DDS requester using the callback endpoint.

3.2 DDS workflow

Figure 3 shows an example DDS workflow. A document updated on one NSA gets propagated throughout the GDS via NSA peering relationships, so that in the end, all peer NSAs within the space have an accurate version of each document within the GDS. In this example, the DDS requester issues an update (e.g version 1.2) to a document sourced on NSA A by using the *updateDocument()* operation. NSA A updates the local document space with the new version of the document, and looks through its subscription list to see if there are any NSAs interested in the document. In this case, NSA B has registered for events on all documents within NSA A. NSA A issues a notification to NSA B with the updated document version 1.2. Similarly, NSA B will update its local document space and issue update notifications to NSA C and D who are also registered with NSA B for events on all documents.

In this example, NSA D will receive update notifications for document version 1.2 from both NSA B and NSA C, however, NSA D will see that the document version for the two different notifications is identical, and discard the duplicate. NSA D then issues a notification to NSA E, which has registered for events on all documents within NSA D. NSA E updates its local document space, and since there are no further NSAs to update, the flow for this update completes. It is important to note that an NSA does not propagate a document notification event back to the NSA from which it was originally received, as this NSA would just discard the update.

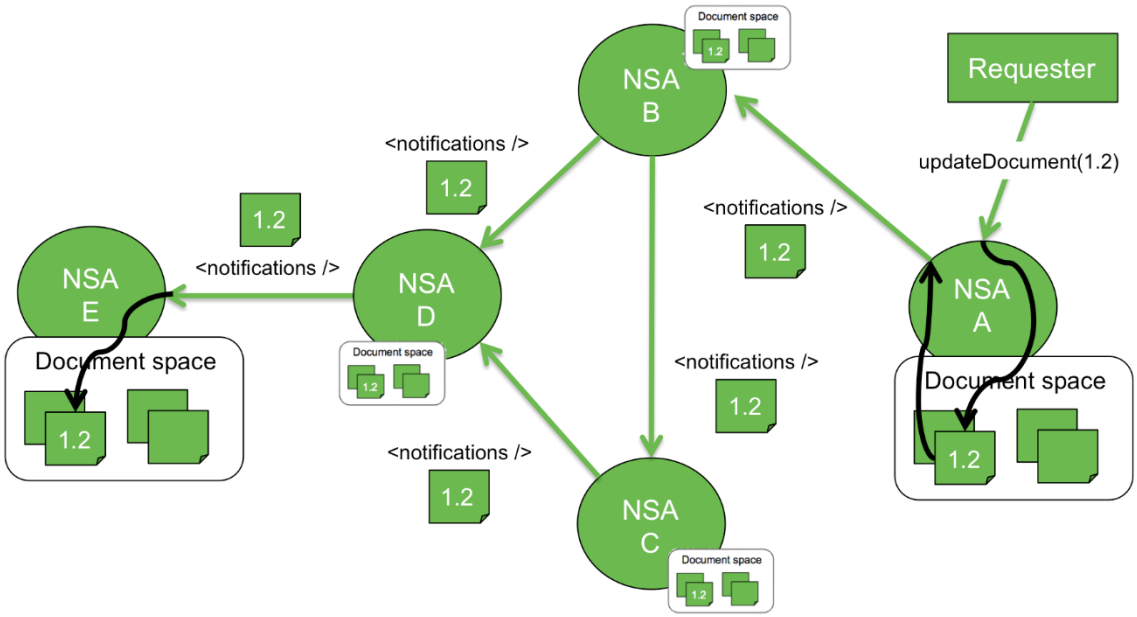

Figure 3 – Document propagation through space.

Additional operations, and more details on the document propagation mechanism are described in more detail in the coming sections.

4 DDS Documents

This section forms a normative part of this recommendation.

A document within the GDS can contain any information that needs to be distributed to all peers participating in the Document Distribution Service. A document is enclosed in meta-data within the GDS space to allow for identification and maintenance. The original document content and annotated meta-data are propagated untouched throughout the GDS.

A document's meta-data entry MUST include the following attributes:

nsa The source NSA associated with the generation and management of the document within the network. This is assumed to be the NSA to which the document relates, however, there may be situations such as proxy publishing where this assumption is not true.

For example, if the document being generated is the NSA Description Document for NSA "*urn:ogf:network:example.com:2013:nsa:vixen*", then the *nsa* element should contain the NSA identifier "*urn:ogf:network:example.com:2013:nsa:vixen*".

type	The unique string identifying the type of this document. A document type is defined by the type and release of a data document. For example, NSI Topology version 1.0 and a NSI Topology version 2.0 would be considered two different document types: • vnd.ogf.nsi.topology.v1+xml • vnd.ogf.nsi.topology.v2+xml The NSA Description Document 1.0 is defined as the type: • vnd.ogf.nsi.nsa.v1+xml Note: the *type* values are currently defined in the NSA description document [OGF NSI-ND]
id	The identifier of the document. This value must be unique in the context of the NSA and type element values.
version	The version of the document, is defined to be the date this version of the document was created. Any updates to the document must be tagged with a new version.
expires	The date this version of the document expires and should be deleted from GDS and any DDS requesters caching the document. More information is provided in Section 5.
signature	An OPTIONAL digital signature of the document content.
content	The document content modeled by this document meta-data.

A document is uniquely identified by the tuple of NSA Identifier (*nsa*), Document Type (*type*), and Document Identifier (*id*). The Document Identifier need only be unique in the context of the NSA Identifier and Document Type. This allows for different types of documents to share the same identifier if they are considered directly related. It also implies that Document Identifiers do not need to be globally unique to be distributed or resolved in the GDS.

The meta-data of each document stored contains a *version* attribute based on the date and time that version of the document was generated. As each new version is added to the space, it replaces the existing version and is propagated to all interested peers.

Meta-data also contains an '*expires'* attribute indicating when the document is no longer valid. Any DDS requesters caching a document that has expired MUST consider the information invalid and discard the document. An NSA within the space MAY keep the expired document for a period of time to guarantee all peers (both polling and subscriptions) have had time to receive the document after it has expired to cover the delete race condition described later in this document.

A document MAY also be digitally signed, generating a *signature* that can be associated with the document within the space. Clients of the space can use the *signature* to verify the originator and content of the document. It is RECOMMENDED that the document being signed includes within its document content a duplicate of the *identifier*, *version*, and *expires* attributes so these values can also be digitally signed and verified if needed.

An NSA MUST not modify the content of a document before propagating on to a peer unless that NSA is the owner of the document.

Section 11 of this document describes a formal specification of a REST-base profile for the DDS protocol through the use of HTTP and XML. A formal XML Schema Definition for this REST-based profile is provided in Section 22 – Appendix IV. Here is an example of an XML instance

from this profile for the meta-data of an NSI Description Document (`vnd.ogf.nsi.nsa.v1+xml`) describing the NSA "`urn:ogf:network:es.net:2013:nsa`".

```
<dds:document xmlns:dds="http://schemas.ogf.org/nsi/2014/02/discovery/types"
        id="urn:ogf:network:es.net:2013:nsa"
        version="2015-12-22T23:44:26.543Z" expires="2016-02-20T23:44:26.543Z">
    <nsa>urn:ogf:network:es.net:2013:nsa</nsa>
    <type>vnd.ogf.nsi.nsa.v1+xml</type>
    <signature contentType="application/x-gzip" contentTransferEncoding="base64">
        H4sIAAAAAAAAAI2V15KrSBKG7/UUij6XbTACmY1WTxROoBYgPOgOU8IbARLm6Qe1dns7zsxEzAV
        QkZH15/9lGd7/6PNsfoN1E5fF9gl7Q5/msPDLIC7C7ZOhc6/rpz8+3rU4LNz2WsP5lF4026eoba
        v/IEjXdW/d4q2sQwRHURRBN8iUEDRx+OvpMQsGQnEuP95ptyiL2HezeHTbqZYI26gM5iALyzpuo
        /zvJHX1roohKku/TrKvPkYUr/cIusDIX9Y0jS7zHBZt84T88PhvpH93WzfuaxO52F1IhWdYT02A
        c0MV/sHX3U7TDhlsIgjbiZWJQ9i0/7I0hqDEXWIq8muqSmL4ve5DwnSzK/ywCR+TOcNsNknbsnU
        95CMwG3zUtbV8CNtC14qFVi6qNvaA0vB1kl/2GYiqLiVuuMicmqgkisWhzqWdOru5h/WRVEm622
        7fkZ9l3pFv2Gn8c72+u/lI9NQgtoeLLvIuInHyhvRvnimcd6m10gPrmnmQ5TnsZq+XIXkoRumUC
        p5V3SK0MoSwJUqZaUikslm/M9ez01pR6RX/eXTlCxk3zmafFunFsOT0tHcvijlGCssiTFuMGo+K
        IYcsP/vKXWdakZ7YPtTLJUAM8tpcTKGb4UTs7Bt252s4UoS1nDLl9oHyw/77JxweXDaJbhi3dR8
        j7eol0G8lN4cf2Bv+tibQNwxbkMTmDXubnu0vbImel8ESWy6W2Ipc+ktytSHQ6b3A4RJOX+KFlr
        b7MirmouuDawaHF9nYsgWsw2Gu+fG9t81cgm1X1umLvDW0OQMrt27vG3denueP1JfDVm5bt3NfN
        H0rSy/0lgbvyO8ev0zTsG7j83SWWvghCgLdjzQNEkDTCt32En89wDVVjUCiwvQSpfFu06EUUAwO
        MPQUZw8iSHcAM1gqEmkFZXt6BHsqlMwZBUIdpBtZVNYdoziMqSg8252OAa9GAc72AgPgPZECpc5
        hpi6waubvuCHYYZlXKKGHO6FqkZmfB+RM1EDHh18iB7YPOC83h9MiDQ1cqk4WmZxsMdRzEw3wze
        COrCVSypcr0IuSlm9KL6Ykx5KowOqzGdRZKFLdjtYuO03wFozC7oHqHD2LSxzbbE52BibB61QEF
        SnCZnQWFRljkHWhE5N0FM3yHluIjNjPZOY72IWu/81PAWfClyhRaTr6gS+w3f6oJ+xRBOijZ724
        0xcq6thBdCec/Rfxc0I0jYFiTzYV+YXaToiosNvkAmteT/a+mAjCydmXCN2L9v/6JHBS4lrm1cH
        NcSawZBbwwc3Pm55LgPFwJeqMUd3cHRHqE67yhZv2fAL8nytMK6zF4Fjk4Bzm7UxypuzM8ZBPgt
        OyF3/ZDiwHgEwlCuhCJ6VDhwVtLmyk23iu6UYbPCaw4quWebRB29Vh9plQbUKo+7Rxrgs+WNXAh
        NFR/5TKZ+9661XG3F8CBpqtanFpE7uEAzdCrGRMJz4PxmF0hVt18wXOKA/O7OzjikaMjV6esxra
        EhY9K7XurJLPoShGzjgltj/0HdopBF8hYlFg8kkjKyONIC/HLOwQlAYdC4AOpNlfdvmEFSqgr5c
        bROm68thN/4h9dl5VC9nSwfpcRiKy2qhRLUjZekCPHCsk6S1ZsZUzyxXuOEpXOor4IMLjuBae10
        CI8wrnr1lqupZHCSjxrAhnyVSvPkP5ZxOKAjZc2l0ndkqkS/vF3l0g3K2brQdG6bXLskxwfE8PG
        jI10tit/KgueYov+TgU3ShrbDq8X8u/n+xH5HFVId/X1/8vto8/AcAmLXe7BwAA
    </signature>
    <content contentType="application/x-gzip" contentTransferEncoding="base64">
        H4sIAAAAAAAAAH1UXW+bMBT9K4jXCWxMkibI0HXVNvVlm5Zsk/bmGkOsgo1sQ9t/v4shH020PkS
        53HvOuT72tentS9sEgzBWapWHSYzDQCiuS6nqPPy1+xKtw9uCKptmyrIAwMpClObh3rkuQ8jyvW
        iZjXVdxdrUSFmJCE4WCBNUSss1aL9CloVHMsnD3qhMCldlHTOstRmUoJANnJkyWvhFvHTSCJuHI
        LaKMIkI3pE0WywysoqXi/RvGMhyEoLWmRLuWZunTNgYwgxIfsHhyRqkllECOuRCB9yxVhSft0AM
        vm/v735ug/7HHUU+Ta2u3DMz4vckVIDBRyPLWkRDQtFllVrHjNtJYP6vIZCOGMrKVqp7rRzjbtx
        mMu3BFPbyEBhZjE57yGR8zcvHxzWOSsZvoiQRabThCYvwGmOCMd+w1QoWP9Pm6KDUGTjaOXbixc
        22o4Pxb9uH4JO3NxE9ZgrPqUYMswa4ADttN9pd4GWS7pLNknibb+vT95H4JNX5MuBLDrLsWXPV9
        4Ss1Bnjfi9V8LXXT1eEA2z+s7Bz40mesIeML9dyEMqrTcXpe4pnBRhw2Zy15gD+OE3aVfMZi86O
        Er095EZz5vysNFrV0vWlKGBOSEyWmKQUnbK0AaCP0pt4fbPaEKgeUiPwIDRP/0NZvHcfYFFHHJX
        KCVMxDkLutRMF67pGTnpoUKW/zzDqMbfxQGI4ezgdYT5YzTqKPIPujaiK8Rmw8A5oC2ZtPPVDYF
        YJPopthRkkh2nyaIrO+laCud6IYFTLw+umRjcihpsYokssPDQXWK7bVvo7pXsXFinGFM2cgnYCb
        ucf6fbBKJmHIPnuRsHDAT8Zsbo20TPML0VHCX+0HlL8A/kqeaU8BQAA
    </content>
</dds:document>
```

This XML *<document>* element represents a signal instance of a document plus associated meta-data. The document being transported is contained in the *<content>* element, and a digital signature for the document is contained in the *<signature>* element. Both the *<content>* and *<signature>* elements are defined as a simple XML string with *contentTransferEncoding* and *contentType* attributes to describe the encoding of the document within this string value based on rules defined in [RFC1341] (sections 5 and 6). The document meta-data *<type>* element identifies the document type itself.

In this example we can see that for the document type "`vnd.ogf.nsi.nsa.v1+xml`" the <signature> and <content> elements contain *contentTransferEncoding* and *contentType* attributes describing additional encoding information. The *contentType* attribute indicates the strings contained in the <signature> and <content> elements are gzipped for compression. In addition, the *contentTransferEncoding* attribute indicates the resulting compressed binary stream is base64 encoded allowing for storage in an XML string. The document type itself identifies the original

type of document stored in the <*content*> element, and the type of signature stored on the <signature> element is based on the document type.

5 Time to Live

This section forms a normative part of this recommendation.

The Document Distribution Service uses the concept of Time To Live (TTL) to set an expiry date on documents exchanged through the DDS. There is no explicit delete operation within the DDS, so the TTL mechanism will ensure old documents eventually expire and are purged from the GDS. This section forms a normative part of this recommendation. The three primary use cases for this feature are:

- An NSA has had a Network removed from its configuration, resulting in the removal of a Topology Document; however, the associated Topology Document was previously announced into the GDS.

- A Network name change has occurred, resulting in a new Topology Document being created and announced into the GDS. This new document has a different unique identifier in the GDS than the Topology Document under the old Network name. As a result, the previously announced document will not be refreshed when the new one is announced, resulting in a stale Topology Document within the GDS. When the TTL on the old Topology Document is reached, all NSA holding a copy will purge it from the GDS.

- An NSA is removed from the Service Plane resulting in the removal of associated Networks from the Data Plane; however, Topology Documents associated with the NSA's Networks were announced into the GDS that are now invalid. When the TTL on the document is reached, all NSAs holding a copy will purge it from the GDS.

In all scenarios, when the TTL on the document is reached, all NSAs holding a copy will purge it from their local DS instance. This will guarantee that the GDS will eventually return to an accurate and consistent state. In the case where the NSA knows a document needs to be deleted, it MUST perform an update on the document, issuing a new version with the *expires* time set to a short period in the future. This update will propagate through the GDS and expire the document at the specified time instead of the original time. The method of removal of the document within an NSA after *expires* time is implementation specific.

An NSA MUST provide an *expires* time with each document published.

Enforcement of *expires* time MUST be based off of a network-synchronized clock.

The *expires* time SHOULD be a reasonable value computed based on the rate of expected change on the document.

6 Subscriptions

This section forms a normative part of this recommendation.

To help support a more dynamic document distribution environment a publish/subscribe model is defined. A provider NSA allows DDS requesters to subscribe to document events by specifying filters, that when matched, will generate document notifications to the subscriber. A DDS requester can also publish documents into a specific provider's document space based on local security policies, which can then result in notification events to subscribed requesters if their

registered filters match the event. For example a uPA may want to publish its documents into an associated aggregators document space.

Each DDS provider also participates in the GDS as a DDS requester, subscribing to document events on peer DDS for any document sourced by other DDS within the GDS. Through this subscription mechanism the DDS requester can dynamically build a global view of the document space without the need to perform document-polling operations on all peer DDS providers.

A subscription entry on a DDS provider is composed of the following attributes:

id	The DDS provider assigned subscription identifier that uniquely identifies the subscription in the context of the provider.
version	The version of the subscription. Indicates the last time the subscription was modified by the DDS requester.
requesterId	The identifier of the DDS requester that created the subscription. A DDS requester agent associated with an NSA should use the NSA's unique identifier for the requesterId. DDSes that are not directly associated with an NSA should utilize a unique identifier following similar name rules as NSA identifiers.
callback	The protocol endpoint on the DDS requester that will receive the notifications delivered for this subscription.
filter	The OPTIONAL filter criteria to apply to document events to determine if a notification should be sent to the DDS requester.

The following is an example subscription request using the formal XML Schema Definition defined in Section 22 – Appendix IV. The NSA "`urn:ogf:network:example.com:2013:nsa:dasher`" is registering a subscription with NSA "`urn:ogf:network:example.com:2013:nsa:dancer`" for all document related events. Notification events will be delivered to the notification endpoint "`http://dasher.example.com/discovery/callback`".

```
<dds:subscriptionRequest xmlns:dds="http://schemas.ogf.org/nsi/2014/02/discovery/types">
    <requesterId>urn:ogf:network:example.com:2013:nsa:dasher</requesterId>
    <callback>http://dasher.example.com/discovery/callback</callback>
    <filter>
        <include>
            <event>All</event>
        </include>
    </filter>
</dds:subscriptionRequest>
```

The response from NSA "`urn:ogf:network:example.com:2013:nsa:dancer`" contains the newly created subscription contained within the DDS service.

```
<dds:subscription xmlns:dds="http://schemas.ogf.org/nsi/2014/02/discovery/types"
    id="1fcca8fb-e33f-46f6-8085-8dbf1a2b346f"
    href="http://dancer.example.com:8401/dds/subscriptions/1fcca8fb-e33f-46f6-8085-
8dbf1a2b346f"
    version="2015-12-08T17:33:49.434-05:00">
    <requesterId>urn:ogf:network:example.com:2013:nsa:dasher</requesterId>
    <callback>http://dasher.example.com/discovery/callback</callback>
    <filter>
        <include>
            <event>All</event>
        </include>
    </filter>
</dds:subscription>
```

A document event that matches the supplied filter will generate notifications that will be delivered to the DDS requester's protocol endpoint specified in the *callback* attribute. Only document events matching the filter criteria will generate a notification event to the subscriber. All other events will be discarded.

Subscription filters allow a subscriber to control the content delivered to their registered notification endpoint. A subscription request without a filter will result in a valid subscription that will match no document events. This can be used to create this initial subscription shell, which can later be modified to add filter criteria as needed.

The filter supports basic criteria:

include – Include notifications matching these criteria.

exclude - Exclude the notifications matching these criteria.

The *include* element specifies the document event match criteria to include, while the *exclude* element specifies those to specifically exclude. The *include* element will be evaluated first, before the *exclude* element. In other words, the *include* is applied to the full documented set producing a bounded output set. The *exclude* then is applied to this bounded set. Each of the *include* and *exclude* elements are composed of:

event – The type of document event that will generate a notification. Currently only three events are supported (***All***, ***New***, ***Updated***). At least one of event criteria must be supplied. The default event criteria is ***All***.

or – Any document matching any of the supplied *nsa*, document *type*, or document *id* values.

and - Any document matching all of the supplied *nsa*, document *type*, or document *id* values.

The following filter subscribes for all document events (***All***) for all discovered documents:

```
<filter>
    <include>
        <event>All</event>
    </include>
</filter>
```

The filter shown above describes the minimum filter criteria for an Aggregator NSA. This filter allows the aggregator to receive all document events from a peer NSA's DDS provider, building a complete view of documents discovered within the GDS. Multiple peers could deliver the same document events, however the aggregator should discard any duplicates. An aggregator receiving duplicate events may decide to modify the filter on a DDS provider to avoid receiving multiple copies of the same document. The following is an example of a filter where the subscriber is still registered for all events, however, it has applied an exclude criteria to stop documents issued by NSA "*urn:ogf:network:example.com:2013:nsa:dasher*" from being sent to the subscriber endpoint:

```
<filter>
    <include>
        <event>All</event>
    </include>
    <exclude>
```

```
            <event>All</event>
            <or><nsa>urn:ogf:network:example.com:2013:nsa:dasher</nsa></or>
        </exclude>
    </filter>
```

An alternative strategy for an aggregator is to initially subscribe to only new document events for its peers, expanding the filter by including individual documents, or documents from specific NSA in the filter as they are first discovered. Using this strategy, the subscribing NSA will only need to update a single subscription to start receiving document updates, instead of excluding from multiple peers as in the previous example.

The initial subscription filter subscribes to new (***New***) document events only for all discovered documents:

```
    <filter>
        <include>
            <event>New</event>
        </include>
    </filter>
```

As new document events arrive, the first peer to report the event can be the peer who is configured to deliver future events for that document to the subscriber. The edited filter would still subscribe to all new document events (***New***), however, we add updates (***Updated***) document events for any documents provided by NSA "*urn:ogf:network:example.com:2013:nsa:vixen*" or "*urn:ogf:network:example.com:2013:nsa:prancer*":

```
<filter>
    <include>
        <event>New</event>
    </include>
    <include>
        <event>Updated</event>
        <or>
            <nsa>urn:ogf:network:example.com:2013:nsa:vixen</nsa>
            <nsa>urn:ogf:network:example.com:2013:nsa:prancer</nsa>
        </or>
    </include>
</filter>
```

Filtering on document type is also supported. The following filter subscribes for all document events (***All***) for discovered documents of type "vnd.ogf.nsi.nsa.v1+xml":

```
<filter>
    <include>
        <event>All</event>
        <or><type>vnd.ogf.nsi.nsa.v1+xml</type></or>
    </include>
</filter>
```

In the above example, since there is only a single entry in the *or* conditional, the filter can also be written using an *and* instead, such as the following:

```
<filter>
    <include>
        <event>All</event>
        <and><type>vnd.ogf.nsi.nsa.v1+xml</type></and>
    </include>
</filter>
```

For each *include* and *exclude* filter criteria, the conditionals therein is evaluated sequentially in relation to the *event*. For example, the following filter subscribes for all document events (***All***) for discovered documents of type "vnd.ogf.nsi.nsa.v1+xml" provided by NSA "*urn:ogf:network:example.com:2013:nsa:vixen*", as well as any (all) document events (***All***) from either NSA "*urn:ogf:network:example.com:2013:nsa:prancer*" or "*urn:ogf:network:example.com:2013:nsa:blitzen*"

```
<filter>
    <include>
        <event>All</event>
        <and>
            <type>vnd.ogf.nsi.nsa.v1+xml</type>
            <nsa>urn:ogf:network:example.com:2013:nsa:vixen</nsa>
        </and>
        <or>
            <nsa>urn:ogf:network:example.com:2013:nsa:prancer</nsa>
            <nsa>urn:ogf:network:example.com:2013:nsa:blitzen</nsa>
        </or>
    </include>
</filter>
```

7 Notifications

This section forms a normative part of this recommendation.

When a document event occurs within the GDS each DDS provider evaluates the event against locally registered subscriptions. For each matching subscription the DDS provider generates a notifications event to the subscribed DDS requester's callback endpoint. Multiple document events matching a single DDS requester's subscription can be bundled into a single notifications event if desired by the DDS provider.

A notifications event generated by a DDS provider is composed of the following attributes:

providerId	The identifier of the DDS provider that holds the subscription that generated the notification.
id	The DDS provider assigned subscription identifier that uniquely identifies the subscription in the context of the provider. This is the identifier of the subscription that generated the document notification.
List of [0..n] notification	A list of document notification events.

Each document notification is composed of the following attributes:

discovered	The time within the DDS provider that the document event occured.
event	The type of document event (New, Updated) that generated the notification.
document	The document that generated the notification.

The following is an example notifications event using the formal XML Schema Definition defined in Section 22 – Appendix IV. For this example we use the example subscription from the previous section. The NSA "`urn:ogf:network:example.com:2013:nsa:dasher`" has registered a

subscription with NSA "urn:ogf:network:example.com:2013:nsa:dancer" for all document related events. The following example notifications event on topology document "urn:ogf:network:example.com:2013:topology:northpole" will be delivered to the notification endpoint "http://dasher.example.com/discovery/callback" based on the subscription criteria:

```
<dds:notifications xmlns:dds="http://schemas.ogf.org/nsi/2014/02/discovery/types"
    providerId=" urn:ogf:network:example.com:2013:nsa:dancer"
    id="1fcca8fb-e33f-46f6-8085-8dbf1a2b346f"
    href="http://dancer.example.com:8401/dds/subscriptions/1fcca8fb-e33f-46f6-8085-
8dbf1a2b346f">
    <dds:notification>
        <discovered>2015-12-10T18:20:49.505-05:00</discovered>
        <event>Updated</event>
        <document id="urn:ogf:network:example.com:2013:topology:northpole"
           href="https://dancer.example.com:8401/dds/documents/urn%3Aogf%3Anetwork
                 %3Aexample.com%3A2013%3Ansa%3Adancer/vnd.ogf.nsi.topology.v2%2Bxml
                 /urn%3Aogf%3Anetwork%3Aexample.com%3A2013%3Atopology%3Anorthpole"
            version="2015-12-10T18:20:49.505-05:00"
            expires="2016-12-10T18:20:49.505-05:00">
            <nsa>urn:ogf:network:example.com:2013:nsa:dancer</nsa>
            <type>vnd.ogf.nsi.topology.v2+xml</type>
            <content contentType="application/x-gzip" contentTransferEncoding="base64">
                H4sIAAAAAAAAAO2bXXObSG/wpDr0ECkzqmiTveqZvNTBp7EjYXe8PIRsaaYokRcpLur1++
                ...
            </content>
        </document>
    </dds:notification>
</dds:notifications>
```

8 Formal API definition

This section forms a normative part of this recommendation.

The logical operations supported by the NSI Document Distribution Service are classified into DDS requester and DDS provider interfaces. A DDS provider "provides" access to documents within the GDS, and a DDS requester is "requesting" access to documents within the GDS. As described earlier, an NSA can participate in both the DDS requester and provider roles.

The DDS provider interface for the NSI Document Distribution Service exposes the following logical operations:

Operation	Returns
getDocuments *([nsa], [type], [id], [lastDiscoveredTime])*	*status, a list of [0..n] documents, and [lastDiscoveredTime]*
getLocalDocuments *([type], [id], [lastDiscoveredTime])*	*status, a list of [0..n] documents, and [lastDiscoveredTime]*
getDocument *(nsa, type, id, [lastDiscoveredTime])*	*status, [document], and [lastDiscoveredTime]*
addDocument *(nsa, type, id, version, expires, [signature], contents)*	*status, [document], and [lastDiscoveredTime]*
updateDocument *(nsa, type, id, version, expires, [signature], contents)*	*status, [document], and [lastDiscoveredTime]*
addSubscription *(requesterId, callback, filter)*	*status, [subscription], and [lastModifiedTime]*
editSubscription *(id, requesterId, callback, filter)*	*status, [subscription], and [lastModifiedTime]*

deleteSubscription *(id)*	*status, and [subscription]*
getSubscriptions *([requesterId], [lastModifiedTime])*	*status, list of [0..n] subscription, and [lastModifiedTime]*
getSubscription *(id, [lastModifiedTime])*	*status, [subscription], and [lastModifiedTime]*
getAll *([lastDiscoveredTime])*	*status, list of [0..n] subscription, list of [0..n] documents, list of [0..n] local documents, and [lastDiscoveredTime]*
notificationCallback(*list of [0..n] notifications****)***	*status*

Table 1 – DDS operations.

8.1 API Access Control

Aspects of security for the DDS API are discussed in Section 12 of this specification. Similar to other NSI specifications, the implementation of security on the DDS interface is implementation/deployment specific. At a minimum, a DDS provider should enforce the following access control rules:

1. Notifications MUST only be accepted from trusted "peer" DDS providers for which valid subscriptions have been created. Unsolicited notification MUST be discarded.
2. Addition of new documents and updates to existing documents within a DDS provider MUST be restricted to authorized DDS requesters.
3. Read access (get operations) SHOULD be restricted to only authorized DDS requesters.
4. Creation of subscription-based notifications SHOULD be restricted to authorized DDS requesters.
5. Editing and deletion of subscriptions SHOULD be restricted to the DDS requester associated with the subscription.

8.2 Operations

The following abstract API operations are defined for the DDS. Within this section the term "content" is used to describe the external "document" information being modeled and distributed within the GDS. The term "document" refers to this content encapsulated within DDS meta-data.

getDocuments*([nsa], [type], [id], [summary], [lastDiscoveredTime])*
RETURNS *status, a list of [0..n] document, and [lastDiscoveredTime]*

This operation returns a list of documents and the time of the latest document change on the DDS provider. If no filter parameters are supplied then all documents within the GDS will be returned. The following optional parameters can be supplied, and will be applied using logical AND:

nsa – The source NSA associated with the generation and management of the document within the GDS.

type - The unique string identifying the type of document to return.

id – The identifier of the document to return.

summary - Returns summary results of any documents matching the query criteria. Summary results includes all document meta-data but not the signature or document contents.

lastDiscoveredTime – Provides a time context to the DDS provider requesting all documents that have been discovered, created, or updated since the time specified in this parameter. This allows for an effective polling mechanism by using the latest document change time returned in the previous operation as a filter parameter in the next get document operation to retrieve only those documents that have been discovered (new or updated) since the last invocation of the API.

In response to this operation the following information is returned:

status – A status indication as to whether the operation was successful or failed. For the case of operation failure informative error information must be provided.

list of [0..n] document – A list of documents matching the provided query criteria.

lastDiscoveredTime – An updated time context indicating the most recent time any document has been discovered, created, updated within the DDS.

getLocalDocuments*([type], [id], [summary], [lastDiscoveredTime])*
RETURNS *status, a list of [0..n] document, and [lastDiscoveredTime]*

This operation returns a list of documents associated with the queried DDS provider and the time of the latest document change on that provider. This operation can be considered equivalent to getDocuments() with the *nsa* parameter set to the target DDS provider's identifier. If no filter parameters are supplied then all documents within the space will be returned. The following optional parameters can be supplied, and will be applied using logical AND:

type - The unique string identifying the type of document to return.

id – The identifier of the document to return.

summary - Returns summary results of any documents matching the query criteria. Summary results includes all document meta-data but not the *signature* or document *contents*.

lastDiscoveredTime – Provides a time context to the DDS provider requesting all documents that have been discovered, created, or updated since the time specified in this parameter. This allows for an effective polling mechanism by using the latest document change time returned in the previous operation as a filter parameter in the next get document operation to retrieve only those documents that have been discovered (new or updated) since the last invocation of the API.

In response to this operation the following information is returned:

status – A status indication as to whether the operation was successful or failed. For the case of operation failure informative error information must be provided.

list of [0..n] document – A list of local documents associated with the target DDS provider.

lastDiscoveredTime – An updated time context indicating the most recent time any document has been discovered, created, updated within the DDS.

getDocument*(nsa, type, id, [lastDiscoveredTime])*
RETURNS *status, [document], and [lastDiscoveredTime]*

This operation returns the requested document and the time of the latest change on the document. The following parameters are used to identify the specific document instance and are mandatory:

nsa – The source NSA associated with the generation and management of the document within the GDS.

type - The unique string identifying the type of document to return.

id – The identifier of the document to return.

If the optional filter parameter *lastDiscoveredTime* is provided, then the target document will only be returned if it has been updated since the time specified.

In response to this operation the following information is returned:

status – A status indication as to whether the operation was successful or failed. For the case of operation failure informative error information must be provided.

document – A document matching the provided *nsa, type*, and *id* parameters if one exists.

lastDiscoveredTime – An updated time context indicating the most recent time this document was discovered, created, or updated within the DDS.

addDocument*(nsa, type, id, version, expires, [signature], content)*
RETURNS *status, [document], and [lastDiscoveredTime]*

This operation adds a new document to the space associated with the DDS provider. Once the document has been successfully created on the provider, a copy of the created document is returned, including the *lastDiscoveredTime* indicating the time the document was added. The provider will immediately send ADD notifications to all subscriptions with filter criteria matching the document.

nsa – The source NSA associated with the generation and management of the document within the GDS.

type - The unique string identifying the type of this document.

id – The identifier of the document being added. This value must be unique in the context of the NSA identifier and document type values, otherwise an error will be returned.

version - The version of the document, or more specifically, the date this version of the document was created.

expires - The date this version of the document expires and should be deleted from document space and any requesters caching the document.

signature - An OPTIONAL digital signature of the document contents.

content - The content of the document modeled by this document meta-data.

In response to this operation the following information is returned:

status – A status indication as to whether the operation was successful or failed. For the case of operation failure informative error information must be provided.

document – The new document (content + meta-data) created within the DDS.

lastDiscoveredTime – The time within the DDS provider that this document was created.

updateDocument*(nsa, type, id, version, expires, [signature], content)*
RETURNS *status, [document], and [lastDiscoveredTime]*

This operation updates an existing document within the space associated with the DDS provider. A document can only be updated within the DDS provider that is acting as the source of the document. Any attempt to update a document from a provider other than the source of the document MUST be rejected. The operation returns a copy of the updated document, and the *lastDiscoveredTime* indicating the time of the document update. The DDS provider will immediately send notifications to all subscriptions with filter criteria matching the document.

This operation is also used to delete an existing document from the space associated with the DDS provider. For the delete of a document the DDS requester issues a new document version with an *expire* time set to a reasonably short period in the future. This updated document propagates through the space to each NSA, updating the previous version to have the immediate expire time. All NSA receiving the document will then have an expired version.

nsa – The source NSA associated with the generation and management of the document within the GDS.

type - The unique string identifying the type of this document.

id – The identifier of the document. This value must be unique in the context of the NSA and type values.

version - The version of the document, or more specifically, the date this version of the document was created. Any updates to the document MUST be tagged with a new version.

expires - The date this version of the document expires and should be deleted from document space and any requesters caching the document.

signature - An OPTIONAL digital signature of the document contents.

content - The content of the document modeled by this document meta-data.

In response to this operation the following information is returned:

status – A status indication as to whether the operation was successful or failed. For the case of operation failure informative error information must be provided.

document – The updated document (content + meta-data) from within the DDS.

lastDiscoveredTime – The time within the DDS provider that this document was updated.

addSubscription*(requesterId, callback, filter)*
RETURNS *status, [subscription], and [lastModifiedTime]*

This operation subscribes a DDS requester for document event notifications based on the supplied filter. Notifications will be delivered to the DDS requester's protocol endpoint specified in the *callback* parameter. This operation returns the newly created subscription including the DDS provider generated subscription *id*, and the *lastModifiedTime* indicating the time the subscription was created.

Once a subscription has been successfully created on the DDS provider, the provider will immediately send notifications for all documents matching the filter criteria excluding the event filter (In this case it consider that the event filter is set to ***All***). This allows a DDS requester to initialize its local cache by getting a complete list of existing documents they are interested in monitoring. For example, if the event filter had been set to ***New*** for all documents, then this initialization behavior will send all matching documents as if they were just discovered.

requesterId - The identifier that the DDS requester would like to use for unique identification. An NSA must use its unique NSA identifier for the *requesterId*.

callback – The DDS requester's endpoint that will receive the notifications delivered for this subscription.

filter - The filter criteria to apply to document events to determine if a notification should be sent to the DDS requester.

In response to this operation the following information is returned:

status – A status indication as to whether the operation was successful or failed. For the case of operation failure informative error information must be provided.

subscription – The created subscription from within the DDS that will contain the unique subscription identifier.

lastModifiedTime – The time within the DDS provider that this subscription was created.

editSubscription*(id, requesterId, callback, filter)*
RETURNS *status, [subscription], and [lastModifiedTime]*

This operation allows a DDS requester to edit an existing subscription subject to access policies. Once a subscription has been successfully edited on the DDS provider, the provider will immediately send notifications for all documents matching the filter criteria excluding the event filter (consider the event filter is set to ***All***). This operation returns the updated subscription and the *lastModifiedTime* indicating the time the subscription was updated.

id – The DDS provider assigned subscription identifier returned by the *addSubscription*() operation.

requesterId - The identifier the DDS requester would like to use for unique identification. An NSA must use its unique NSA identifier for *requesterId*.

callback – The DDS requester's protocol endpoint that will receive the notifications delivered for this subscription.

filter - The filter criteria to apply to document events to determine if a notification should be sent to the DDS requester.

In response to this operation the following information is returned:

status – A status indication as to whether the operation was successful or failed. For the case of operation failure informative error information must be provided.

subscription – The edited subscription from within the DDS.

lastModifiedTime – The time within the DDS provider that this subscription was edited.

deleteSubscription*(id)* ***RETURNS*** *status, and [subscription]*

This operation deletes the subscription associated with *id* from the DDS provider subject to access policies. The deleted subscription is returned.

id – The DDS provider assigned subscription identifier returned by the *addSubscription*() operation.

In response to this operation the following information is returned:

status – A status indication as to whether the operation was successful or failed. For the case of operation failure informative error information must be provided.

subscription – The deleted subscription from within the DDS.

getSubscriptions*([requesterId], [lastModifiedTime])*
RETURNS *status, list of [0..n] subscription, and [lastModifiedTime]*

This operation returns a list of subscriptions and the time of the latest subscription change on the DDS provider. If no filter parameters are supplied then all subscriptions on the provider will be returned subject to access policies. The following optional parameters can be supplied, and will be applied using logical AND:

requesterId – Return only subscriptions for this unique requester identifier.

lastModfiedTime – Provides a time context to the DDS provider requesting all subscriptions that have been created or modified since the time specified in this parameter.

In response to this operation the following information is returned:

status – A status indication as to whether the operation was successful or failed. For the case of operation failure informative error information must be provided.

list of [0..n] subscription – A list of subscriptions within the target DDS provider matching the query parameters.

lastModifiedTime – Time context indicating the most recent time a subscription within the DDS provider has been created or updated.

getSubscription*(id, [lastModifiedTime])*
RETURNS *status, [subscription], and [lastModifiedTime]*

This operation returns a single subscription identified by the *id* parameter and the time this subscription was last modified subject to access policies.

id – The DDS provider assigned subscription identifier returned by the *addSubscription*() operation.

LastModifiedTime – This OPTIONAL parameter provides a time context to the DDS provider NSA requesting the subscription only be returned if it has been modified since the time specified in this parameter.

In response to this operation the following information is returned:

status – A status indication as to whether the operation was successful or failed. For the case of operation failure informative error information must be provided.

subscription – The subscription within the target DDS provider matching the query parameters.

lastModifiedTime – Time context indicating the most recent time this subscription was created or updated.

getAll*([lastDiscoveredTime])* ***RETURNS*** *status, list of [0..n] subscription, list of [0..n] document, list of [0..n] local document, and [lastDiscoveredTime]*

This operation returns a collection of subscriptions, documents, and local documents discovered since *lastDiscoveredTime (*treating *lastDiscoveredTime* as *lastModifiedTime* in the case of subscriptions*)* subject to access policies. The time of the last discovered/modified element is also returned.

lastDiscoveredTime – This OPTIONAL parameter provides a time context to the DDS provider NSA requesting the subscriptions and documents only be returned if it has been modified since the time specified in this parameter.

In response to this operation the following information is returned:

status – A status indication as to whether the operation was successful or failed. For the case of operation failure informative error information must be provided.

list of [0..n] subscription – A list of subscriptions within the target DDS provider matching the query parameters.

list of [0..n] document – A list of documents within the target DDS provider matching the query parameters.

list of [0..n] local document – A list of local documents within the target DDS provider matching the query parameters.

lastDiscoveredTime – Time context indicating the most recent time a document or subscription within the DDS provider has been discovered, created, or updated.

notificationCallback*(list of [0..n] notification)* ***RETURNS*** *status*

The DDS requester exposes this API method to receive notifications from a DDS provider matching a previously registered active subscription.

list of [0..n] notification – A list of document notifications matching a previously active registered subscription.

In response to this callback the DDS requester returns the following information:

status – A status indication as to whether the operation was successful or failed. For the case of operation failure informative error information must be provided.

9 NSA Bootstrap Procedure

This section forms a normative part of this recommendation.

One of the important uses of the NSI Document Distribution Service is the simplification of NSA provisioning through dynamic retrieval of the NSA Description Document. Utilizing the meta-data contained in a peer NSA's Description Document it is possible to programmatically configure most of the information required to bring up the NSI suite of protocols. This section describes a basic procedure that can be followed that is compliant with the NSI 2.0 protocol suite.

To bring up NSI communication between two peer NSAs, the NSA administrators must configure a local peering relationship:

1. Exchange TLS certificates and NSI Document Distribution Service endpoints with the system administrator of the peer DDS agent.
2. Provision a remote peer TLS certificate in the local NSA's local trust store to enable transport communications.
3. Provision a peer certificate DN in NSA authorization module if additional application level validation is desired.
4. Provision the NSI Document Distribution Service URL in NSA for bootstrap procedure.

On NSA peering initialization:

1. The local NSA connects to Document Distribution Service on a peer NSA using the configured endpoint and TLS as a transport.
2. The local NSA performs a ***getLocalDocuments()*** operation to retrieve the peer NSA's Description Document and any other documents associated with the peer NSA.
3. The NSA identifier of the peer NSA and all associated Networks are now known.
4. For each NSI service on local NSA, determine highest common interface version described in the peer NSA's Description Document. The decision about the version of the interface to use is made by the NSA in the RA role
5. Utilize interfaces and feature information as need.

For uRA (requester only NSA) this procedure is optional if the administrator would rather manually provision the required information.

10 Peer flooding and version sequencing

This section forms a normative part of this recommendation.

Due to the selective connectivity between NSAs and the transfer latency between any pair, it is important that the NSI Document Distribution Service facilitate convergence of information over all the DDS providers. Figure 8 shows an example of such a scenario.

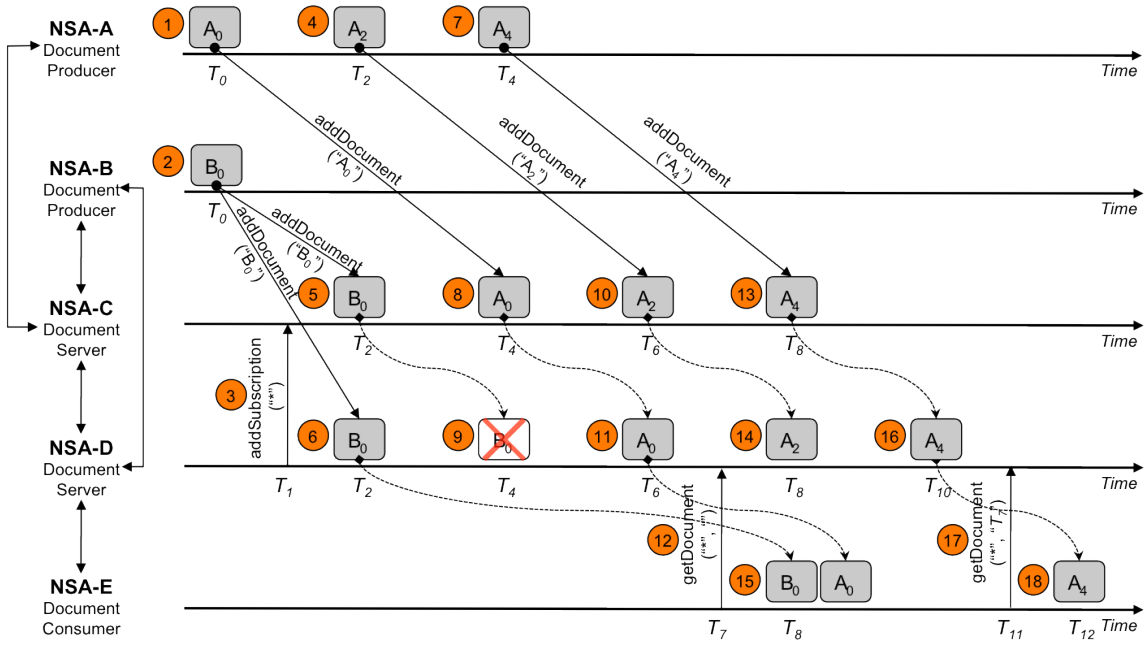

Figure 4 – Document flooding.

1. At time=T_0, NSA-A (a uPA) produces a document A_0 (i.e. document "*A*", version "*0*") and pushes it to NSA-C (an AG)
2. At time=T_0, NSA-B (a uPA) produces a document B_0 and pushes it to NSA-C and NSA-D (an AG)
3. At time=T_1, NSA-D sends a subscribe to NSA-C for all documents
4. At time=T_2, NSA-A produces a document A_2 and pushes it to NSA-C
5. At time=T_2, NSA-C receives document B_0 from NSA-B and sends a copy to NSA-D (base on the subscribe request time=T_1)
6. At time=T_2, NSA-D receives document B_0 from NSA-B
7. At time=T_4, NSA-A produces a document A_4 and pushes it to NSA-C
8. At time=T_4, NSA-C receives document A_0 from NSA-A and sends a copy to NSA-D
9. At time=T_4, NSA-D receives document B_0 from NSA-C (base on the subscribe request at time=T_1) but discards it because it already has a copy of document B_0 (from NSA-B received at time=T_2)
10. At time=T_6, NSA-C receives document A_2 (which deprecates A_0) from NSA-A and sends a copy to NSA-D
11. At time=T_6, NSA-D receives document A_0 from NSA-C
12. At time=T_7, NSA-E (a uPA) sends a request to NSA-D for all documents that it knows about
13. At time=T_8, NSA-C receives document A_4 (which deprecates A_2) from NSA-A and sends a copy to NSA-D
14. At time=T_8, NSA-D receives document A_2 (which deprecates A_0) from NSA-C

15. At time=T_8, NSA-E receives document B_0 and A_0 from NSA-D
16. At time=T_{10}, NSA-D receives document A_4 (which deprecates A_2) from NSA-C
17. At time=T_{11}, NSA-E sends a request to NSA-D for all new documents that it (NSA-D) has learned about since time=T_7
18. At time=T_{12}, NSA-E receives document A_4 (which deprecates A_0) from NSA-D

11 REST-based Protocol Profile

This section forms a normative part of this recommendation.
The NSI Document Distribution Service is implemented using a REST-based design pattern to create an HTTP based web service. This provides a lighter weight design than the NSI CS SOAP based specification, and simplifies the overall protocol stack for a discovery service that needs to be as simple as possible. This section provides a mapping from the abstract Document Distribution Service operations to concrete HTTP binding for the protocol. More information on the REST design pattern and best practices can be found in [FIELDING] and [RICH].

Table 2 describes the basic resources modeled in the Document Distribution Service REST API and the HTTP methods supported on the resources. As a standard design pattern, this protocol uses the HTTP GET method of retrieving and querying resources, the POST method for creating new instances of resources, the PUT method for updating a resource, and the DELETE method for deleting a resource.

Resource	Methods	Description
collection	GET	This root resource contains a collection of zero or more subscriptions and documents held within the NSA.
subscriptions	GET, POST	This resource represents a group of zero or more subscription instances.
subscription	GET, PUT, DELETE	This resource represents a single subscription instance.
documents	GET, POST	This resource represents a group of zero or more document instances.
document	GET, PUT	This resource represents a single document instance.
local	GET	This resource represents a group of zero or more document instances associated with the local NSA.

Table 2 – Resources.

Table 3 describes the URI template mappings for the resources previously described.

Resource	URI	Description
collection	/	Using root URI with a GET operation will return a collection of zero or more subscriptions and documents held within the NSA.
subscriptions	/subscriptions	Using this URI with a GET operation will return a group of zero or more subscription instances. Using this URI with a POST operation will create a new subscription with the supplied criteria.
subscription	/subscriptions/{subscriptionId}	Use this URI template to access a single subscription instance based on subscription identifier. Using a GET operation will get the subscription identified by {*subscriptionId}*. Using a PUT operation will update the subscription identified by *{subscriptionId*} with the values supplied in the PUT body (*subscriptionRequest* element). Using a DELETE operation will remove the subscription identified by {*subscriptionId}*.
documents	/documents	Using this URI with a GET operation will return a group of zero or more document instances. Using this URI with a POST operation will create a new document with the supplied values (*document* element).
documents	/documents/{nsaId}	Use this URI template to access a list of document instances associated with an NSA identifier. Using this URI with a GET operation will return a group of zero or more document instances associated with the NSA identified by {*nsaId}*.
documents	/documents/{nsaId}/{type}	Use this URI template to access a list of document instances associated with an NSA identifier and specific document type. Using this URI with a GET operation will return a group of zero or more document instances of the document type *{type}* associated with the NSA identified by {*nsaId}.*
document	/documents/{nsaId}/{type}/{id}	Use this URI template to access a single document instance associated with an NSA identifier, document type, and document identifier. Using this URI with a GET operation will return a single document instance (*document* element) associated with the document identifier *{id},* the type *{type}, and* the NSA identified by {*nsaId}*. Using a PUT operation will update the document identified by *{id*} with the values supplied in the PUT body (*document* element). This can only be done by an authorized entity. This is the mechanism to provide an updated version of the document.
local	/local	Using this URI with a GET operation will return a group of zero or more document instances associated with the local NSA.

Table 3 – URIs.

11.1 Content Encodings

The NSI Document Distribution Service Protocol mappings utilize custom MIME types carried in the *Content-Type* and *Accept* HTTP header parameters to identify the version of the resources carried in the HTTP body. Resources are intentionally defined to be generic enough that they should not need to be up-versioned. In the case that the protocol needs to identify a change in format of the resource, a new MIME type can be created.

On the HTTP POST and PUT request the *Content-Type* parameter identifies the version of resource carried in the body of the operation, and the *Accept* parameter identifies the version of resource acceptable on output. The HTTP response contains a *Content-Type* parameter identifying the version of resource contained in the response.

The following string uniquely identifies this version of the document distribution service:

"vnd.ogf.nsi.dds.v1"

The following MIME type is defined to identify the XML content encoding for this specific version of the service:

"application/vnd.ogf.nsi.dds.v1+xml"

The default content encoding for XML MUST also be supported for the newest version of the service:

"application/xml"

Further content encodings, including JSON, MAY be specified in a future version of the standard as needed.

11.2 Operations

This section describes the mappings of the abstract Document Distribution Service API operations to the physical REST-based service.

11.2.1 getDocuments

Method: GET /documents

This operation returns all document instances discovered within the document space, or a subset of documents based on supplied query parameters. Zero or more document instances are returned in the *documents* element. Any results returned are based on the permissions of the DDS requester.

The URI template *"/documents/{nsa}/{type}"* can be used as an alternative to, or in conjunction with, the use of query parameters. Performing a GET on *"/documents/{nsa}/"* returns all documents associated with the specified NSA. Performing a GET on *"/documents/{nsa}/{type}"* returns all documents of *{type}* from the specified NSA.

Header Parameters

The following header parameters are supported for the documents resource.

Parameter	Value	Description
Accept	String	Identifies the content type encoding requested for

		the returned results. Must be a content type supported by the protocol.
If-Modified-Since	RFC1123 date string	Constrains the GET request to return only those documents that have been created or updated since the time specified in this parameter. If the query on the documents resource would have returned results, but applying these criteria results in an empty set of documents, a 304 (not modified) response will be returned without any message-body.

Query Parameters

The following query parameters are supported for the subscriptions resource. Query parameters are applied with a logical AND when there is more than one.

Parameter	Value	Description
id	String	Return all document resources containing the specified *Id*.
nsa	String	Returns all document resources containing the specified *nsa* identifier. Cannot be used if the {nsa} URI component is provided.
type	String	Returns all document resources containing the specified *type*. Cannot be used if the {type} URI component is provided.
summary	True/false	Returns summary results of any documents matching the query criteria. Summary results includes all document meta-data but not the *signature* or document *contents*.

Returns

The following information can be returned in response to the query.

Status Code	Element	Description
200	*documents*	Returns the *documents* element containing all document resources matching the query. If no documents match the query, then an empty *documents* element is returned.
304	N/A	Successful operation where there were no changes to any document resource given the *If-Modified-Since* criteria. Returns no message body.
400	*error*	Returned if a DDS requester specifies an invalid request. An *error* element will be included populated with appropriate error information.
401	*error*	Returned if the DDS requester is not authorized to perform the requested operation or access the targeted resource. An *error* element will be included populated with appropriate error information.
500	*error*	Returned if an internal server error occurred during the processing of this request. An *error* element will be included populated with appropriate error information.

Example

The following example shows a valid ***GET*** request on the "*/documents*" resource with a *type* query parameter. The result is a list of *document* resources matching the query parameter after any access control was applied:

```
GET /documents?type=vnd.ogf.nsi.topology.v2+xml HTTP/1.1
Accept: application/vnd.ogf.nsi.dds.v1+xml

HTTP/1.1 200 OK
Date: Mon, 10 Feb 2014 22:12:59 GMT
Content-Length: 648
Last-Modified: Mon, 10 Feb 2014 22:12:05 GMT
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:documents xmlns:tns="http://schemas.ogf.org/nsi/2013/04/discovery/types"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <tns:document id="urn:ogf:network:example.com:2013:network:candycaneforest"
            version="2014-02-10T22:20:58Z" expires="2014-02-11T22:20:58Z">
        <nsa>urn:ogf:network:example.com:2013:nsa:vixen</nsa>
        <type>vnd.ogf.nsi.topology.v2+xml</type>
        <signature>...</signature>
        <content>...</content>
    </tns:document>
    <tns:document id="urn:ogf:network:example.com:2013:network:lincolntunnel"
            version="2014-02-10T22:15:10Z" expires="2014-02-11T22:15:10Z">
        <nsa>urn:ogf:network:example.com:2013:nsa:prancer</nsa>
        <type>vnd.ogf.nsi.topology.v2+xml</type>
        <signature> ... </signature>
        <content> ... </content>
    </tns:document>
</tns:documents>
```

11.2.2 getLocalDocuments

Method: GET /local

A DDS requester can perform a GET operation on the special *"/local"* URI when the DDS requester would like to discover all documents associated with the local NSA. The local NSA returns a *documents* element containing a list of zero or more document instances associated with the local NSA. This operation is equivalent to performing a GET operation on the URI *"/documents/{nsa}"*, however, for *"/local"* the DDS requester is not required to have previous knowledge of the local NSA identifier.

The URI template *"/local/{type}"* can be used as an alternative to, or in conjunction with, the use of query parameters. Performing a GET on *"/local/{type}/"* will return all documents of *{type}* associated with the local NSA.

Header Parameters

The following header parameters are supported for the documents resource.

Parameter	Value	Description
Accept	String	Identifies the content type encoding requested for the returned results. Must be a content type supported by the protocol.
If-Modified-Since	RFC1123 date string	Constrains the GET request to return only those documents that have been created or updated since the time specified in this parameter. If the query on the documents resource would have returned results, but applying these criteria results in an empty set of documents, a 304 (not modified)

response will be returned without any message-body.

Query Parameters

The following query parameters are supported for the subscriptions resource. Query parameters are applied with a logical AND when there is more than one.

Parameter	Value	Description
id	String	Returns all document resources containing the specified *Id*.
type	String	Returns all document resources containing the specified *type*.
summary	True/false	Returns summary results of any documents matching the query criteria. Summary results includes all document meta-data but not the *signature* or document *content*.

Returns

The following information can be returned in response to the query.

Status Code	Element	Description
200	*local*	Returns the *documents* element containing all document resources matching the query. If no documents match the query, then an empty *documents* element is returned.
304	NA	Successful operation where there were no changes to any document resource given the *If-Modified-Since* criteria. Returns no message body.
400	*error*	Returned if a DDS requester specifies an invalid request. An *error* element will be included populated with appropriate error information.
401	*error*	Returned if the DDS requester is not authorized to perform the requested operation or access the targeted resource. An *error* element will be included populated with appropriate error information.
500	*error*	Returned if an internal server error occurred during the processing of this request. An *error* element will be included populated with appropriate error information.

Example

The following example shows a valid ***GET*** request on the "*/local*" resource with a *type* query parameter. The result is a list of *document* resources matching the query parameter after any access control was applied:

```
GET /local?type=vnd.ogf.nsi.topology.v2+xml HTTP/1.1
Accept: application/vnd.ogf.nsi.dds.v1+xml

HTTP/1.1 200 OK
Date: Mon, 10 Feb 2014 22:12:59 GMT
Content-Length: 648
Last-Modified: Mon, 10 Feb 2014 22:12:05 GMT
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:local xmlns:tns="http://schemas.ogf.org/nsi/2013/04/discovery/types"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <tns:document id="urn:ogf:network:example.com:2013:network:candycaneforest"
```

```
            version="2014-02-10T22:20:58Z" expires="2014-02-11T22:20:58Z">
        <nsa>urn:ogf:network:example.com:2013:nsa:vixen</nsa>
        <type>vnd.ogf.nsi.topology.v2+xml</type>
        <signature>...</signature>
        <content>...</content>
    </tns:document>
    <tns:document id="urn:ogf:network:example.com:2013:network:lincolntunnel"
            version="2014-02-10T22:15:10Z" expires="2014-02-11T22:15:10Z">
        <nsa>urn:ogf:network:example.com:2013:nsa:prancer</nsa>
        <type>vnd.ogf.nsi.topology.v2+xml</type>
        <signature> ... </signature>
        <content> ... </content>
    </tns:document>
</tns:local>
```

11.2.3 addDocument

Method: POST /documents

The POST operation on the *"/documents*" resource will create a new document using the information supplied in the *document* element contained in the POST body. A successful operation will return the new document resource. This operation has restricted access for DDS requesters and is made available by the DDS provider based on access control permissions.

Once a document has been successfully created on the DDS provider, the provider will immediately send notifications to all subscriptions with filter criteria matching the document.

Header Parameters

The following header parameters are supported for the request for a new document resource.

Parameter	Value	Description
Content-Type	String	Identifies the content type encoding of the POST body contents. Must be a content type supported by the protocol.
Accept	String	Identifies the content type encoding requested for the returned results. Must be a content type supported by the protocol.

Body Parameters

The POST request must contain the *document* element containing the parameters of the *document* resource to be created.

Parameter	Value	Description
Id	xsd:string	The identifier of the document. This value must be unique in the context of the nsa and type values.
version	xsd:dateTime	The version of the document. Typically the date this version of the document was created. Any updates to the document must be tagged with a new version.
expires	xsd:dateTime	The date this version of the document expires and should be deleted from the NSA (local DS instance) and any DDS requesters caching the document.
Nsa	xsd:anyURI	The source NSA associated with the generation and management of the document.
type	xsd:string	The unique string identifying the type of this document.
signature	ContentType	The OPTIONAL digital signature of the document content.
content	ContentType	The content of the document modeled by this document resource.

Returns

The following information can be returned in response to the POST.

Status Code	Element	Description
201	*document*	Returns a copy of the new document resource created as the result of a successful operation. The HTTP *Location* header field will contain the direct URI reference of the new document resource. It will be structured using the URI template $root/documents/{nsa}/{type}/{id}.
400	*error*	Returned if a DDS requester specifies an invalid request. An *error* element will be included populated with appropriate error information.
401	*error*	Returned if the DDS requester is not authorized to perform the requested operation or access the targeted resource. An *error* element will be included populated with appropriate error information.
409	*error*	A document already exists with the same name (nsa/type/id). An update of an existing document should use the PUT operation.
500	*error*	Returned if an internal server error occurred during the processing of this request. An *error* element will be included populated with appropriate error information.

Example

The following example shows a valid ***POST*** request on the "*/documents*" resource:

```
POST /documents HTTP/1.1
Accept: application/vnd.ogf.nsi.dds.v1+xml
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:document xmlns:tns="http://schemas.ogf.org/nsi/2013/04/discovery/types"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         id="urn:ogf:network:example.com:2013:network:candycaneforest"
        version="2014-02-10T22:20:58Z" expires="2014-02-11T22:20:58Z">
    <nsa>urn:ogf:network:example.com:2013:nsa:vixen</nsa>
    <type>vnd.ogf.nsi.topology.v2+xml</type>
    <signature>...</signature>
    <content>...</content>
</tns:document>

HTTP/1.1 201 Created
Date: Mon, 10 Feb 2014 22:21:59 GMT
Content-Length: 563
Last-Modified: Mon, 10 Feb 2014 22:21:58 GMT
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
Location:
/documents/urn:ogf:network:example.com:2013:nsa:vixen/vnd.ogf.nsi.topology.v2+xml/urn:ogf:
network:example.com:2013:network:candycaneforest
<?xml version="1.0" encoding="UTF-8"?>
<tns:document xmlns:tns="http://schemas.ogf.org/nsi/2013/04/discovery/types"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        id="urn:ogf:network:example.com:2013:network:candycaneforest"
        version="2014-02-10T22:20:58Z" expires="2014-02-11T22:20:58Z">
    <nsa>urn:ogf:network:example.com:2013:nsa:vixen</nsa>
    <type>vnd.ogf.nsi.topology.v2+xml</type>
    <signature>...</signature>
    <content>...</content>
</tns:document>
```

11.2.4 getDocument

Method: GET /documents/{nsa}/{type}/{id}

This operation will return a specific document instance discovered within the document space, subject to access policy, based on the URI template *"/documents/{nsa}/{type}/{id}",* where *{nsa*} is the NSA sourcing the document, *{type}* is the type of document, and *{id}* is the identifier of the specific document. The matching document is returned in a single *document* element.

Header Parameters

The following header parameters are supported for the subscriptions resource.

Parameter	Value	Description
Accept	String	Identifies the content type encoding requested for the returned results. Must be a content type supported by the protocol.
If-Modified-Since	RFC1123 date string	Constrains the GET request to return the matching document only if it has been updated since the time specified in this parameter. If the subscription resource does not meet these criteria, a 304 (not modified) response will be returned without any message-body.

Query Parameters

None.

Returns

The following information can be returned in response to the GET of a subscription.

Status Code	Element	Description
200	*document*	Successful operation returns the document identified by *{nsa}/{type}/{id}* in a *document* element. The *Last-Modified* header parameter will contain the time this document resource was last discovered.
304	NA	Successful operation where there were no changes to the document resource given the *If-Modified-Since* criteria. Returns no message body.
400	*error*	Returned if a DDS requester specifies an invalid request. An *error* element will be included populated with appropriate error information.
401	*error*	Returned if the DDS requester is not authorized to perform the requested operation or access the targeted resource. An *error* element will be included populated with appropriate error information.
404	*error*	Returned if the requested document was not found. An *error* element will be included populated with appropriate error information.
500	*error*	Returned if an internal server error occurred during the processing of this request. An *error* element will be included populated with appropriate error information.

Example

The following example shows a valid ***GET*** request on the document resource identified by *the* URI *"/documents/urn:ogf:network:example.com:2013:nsa:vixen/vnd.ogf.nsi.topology.v2+xml/urn:ogf:network:example.com:2013:network:candycaneforest".* The result is a single *document* resource:

```
GET
/documents/urn:ogf:network:example.com:2013:nsa:vixen/vnd.ogf.nsi.topology.v2+xml/urn:ogf:
network:example.com:2013:network:candycaneforest HTTP/1.1
Accept: application/vnd.ogf.nsi.dds.v1+xml

HTTP/1.1 200 OK
Date: Mon, 10 Feb 2014 22:21:59 GMT
Content-Length: 563
Last-Modified: Mon, 10 Feb 2014 22:21:58 GMT
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:document xmlns:tns="http://schemas.ogf.org/nsi/2013/04/discovery/types"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       id="urn:ogf:network:example.com:2013:network:candycaneforest"
       version="2014-02-10T22:20:58Z" expires="2014-02-11T22:20:58Z">
    <nsa>urn:ogf:network:example.com:2013:nsa:vixen</nsa>
    <type>vnd.ogf.nsi.topology.v2+xml</type>
    <signature>...</signature>
    <content>...</content>
</tns:document>
```

11.2.5 updateDocument

Method: PUT /documents/{nsa}/{type}/{id}

The PUT operation on the *"/documents/{nsa}/{type}/{id}*" resource will allow a DDS requester to edit the document corresponding to the identifier *{id},* subject to access policy, using the information supplied in the *document* element contained in the PUT body. A successful operation will return the modified document and trigger any associated notifications within the NSA.

A document is deleted from the document space by updating it's expire date to a reasonably short period in the future. This updated document will get propagated throughout the document space and then expire, removing it from the space.

Header Parameters

The following header parameters are supported for the request edit a document resource.

Parameter	Value	Description
Content-Type	String	Identifies the content type encoding of the PUT body contents. Must be a content type supported by the protocol.
Accept	String	Identifies the content type encoding requested for the returned results. Must be a content type supported by the protocol.

Body Parameters

The PUT request must contain the *document* element containing the existing parameters of the *document* resource if they were not modified, as well as any new/edited values.

Parameter	Value	Description
id	xsd:string	The identifier of the document. This value must be

		unique in the context of the nsa and type values.
version	xsd:dateTime	The version of the document. Typically the date this version of the document was created. Any updates to the document must be tagged with a new version.
expires	xsd:dateTime	The date this version of the document expires and should be deleted from the NSA (document server) and any DDS requesters caching the document.
nsa	xsd:anyURI	The source NSA associated with the generation and management of the document.
type	xsd:string	The unique string identifying the type of this document.
signature	ContentType	The OPTIONAL digital signature of the document content.
content	ContentType	The content of the document modeled by this document resource.

Returns

The following information can be returned in response to the PUT.

Status Code	Element	Description
200	*document*	Returns a copy of the modified document resource as the result of a successful operation.
400	*error*	Returned if a DDS requester specifies an invalid request. An *error* element will be included populated with appropriate error information.
401	*error*	Returned if the DDS requester is not authorized to perform the requested operation or access the targeted resource. An *error* element will be included populated with appropriate error information.
404	*error*	Returned if the requested document was not found. An *error* element will be included populated with appropriate error information.
500	*error*	Returned if an internal server error occurred during the processing of this request. An *error* element will be included populated with appropriate error information.

Example

The following example shows a valid ***PUT*** request on the document *"/documents/urn:ogf:network:example.com:2013:nsa:vixen/vnd.ogf.nsi.topology.v2+xml/urn:ogf:network:example.com:2013:network:candycaneforest"* with updated version and expire attributes.

```
PUT
/documents/urn:ogf:network:example.com:2013:nsa:vixen/vnd.ogf.nsi.topology.v2+xml/urn:ogf:
network:example.com:2013:network:candycaneforest HTTP/1.1
Accept: application/vnd.ogf.nsi.dds.v1+xml
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:document xmlns:tns="http://schemas.ogf.org/nsi/2013/04/discovery/types"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       id="urn:ogf:network:example.com:2013:network:candycaneforest"
       version="2014-02-12T22:20:58Z" expires="2014-02-13T22:20:58Z">
    <nsa>urn:ogf:network:example.com:2013:nsa:vixen</nsa>
    <type>vnd.ogf.nsi.topology.v2+xml</type>
    <signature>...</signature>
    <content>...</content>
</tns:document>

HTTP/1.1 200 OK
```

```
Date: Mon, 12 Feb 2014 22:20:59 GMT
Content-Length: 563
Last-Modified: Mon, 12 Feb 2014 22:20:58 GMT
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
Location: /
documents/urn:ogf:network:example.com:2013:nsa:vixen/vnd.ogf.nsi.topology.v2+xml/urn:ogf:n
etwork:example.com:2013:network:candycaneforest
<?xml version="1.0" encoding="UTF-8"?>
<tns:document xmlns:tns="http://schemas.ogf.org/nsi/2013/04/discovery/types"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       id="urn:ogf:network:example.com:2013:network:candycaneforest"
       version="2014-02-12T22:20:58Z" expires="2014-02-13T22:20:58Z">
    <nsa>urn:ogf:network:example.com:2013:nsa:vixen</nsa>
    <type>vnd.ogf.nsi.topology.v2+xml</type>
    <signature>...</signature>
    <content>...</content>
</tns:document>
```

11.2.6 getSubscriptions

Method: GET /subscriptions

Return a *subscriptions* element containing a list of zero or more subscription instances based on supplied parameters and permissions of the DDS requester, subject to access policy.

Header Parameters

The following header parameters are supported for the subscriptions resource.

Parameter	Value	Description
Accept	String	Identifies the content type encoding requested for the returned results. Must be a content type supported by the protocol.
If-Modified-Since	RFC1123 date string	Constrains the GET request to return only those subscriptions that have been created or updated since the time specified in this parameter. If the query on the subscriptions resource would have returned results, but applying these criteria results in an empty set of documents, a 304 (not modified) response will be returned without any message-body.

Query Parameters

The following query parameters are supported for the subscriptions resource.

Parameter	Value	Description
requesterId	String	Returns all subscription resources containing the specified *requesterId*.

Returns

The following information can be returned in response to the query.

Status Code	Element	Description
200	*subscriptions*	Returns all subscription resources matching the query in a *subscriptions* element. If no subscriptions match the query, then an empty *subscriptions* element is returned.
304	NA	Successful operation where there were no changes to any

		subscription resources matching the query filter given the *If-Modified-Since* criteria. Returns no message body.
400	*error*	Returned if a DDS requester specifies an invalid request. An *error* element will be included populated with appropriate error information.
401	*error*	Returned if the DDS requester is not authorized to perform the requested operation or access the targeted resource. An *error* element will be included populated with appropriate error information.
500	*error*	Returned if an internal server error occurred during the processing of this request. An *error* element will be included populated with appropriate error information.

Example

The following example shows a valid ***GET*** request on the "*/subscriptions*" resource with a *requesterId* query parameter. The result is a list of *subscription* resources matching the query parameter after any access control is applied:

```
GET /subscriptions?requesterId=urn:ogf:network:example.com:2013:nsa:dasher HTTP/1.1
Accept: application/vnd.ogf.nsi.dds.v1+xml

HTTP/1.1 200 OK
Date: Mon, 10 Feb 2014 22:12:59 GMT
Content-Length: 648
Last-Modified: Mon, 10 Feb 2014 22:12:05 GMT
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:subscriptions xmlns:tns="http://schemas.ogf.org/nsi/2013/04/discovery/types"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <tns:subscription
            id="9e223d413578"
            href="/subscriptions/9e223d413578"
            version="2014-02-10T22:12:05Z">
        <requesterId>urn:ogf:network:example.com:2013:nsa:dasher</requesterId>
        <callback>http://dasher.example.com/discovery/callback</callback>
        <filter>
            <include>
                <event>All</event>
            </include>
        </filter>
    </tns:subscription>
</tns:subscriptions>
```

11.2.7 addSubscription

Method: POST /subscriptions

The POST operation on the *"/subscriptions*" resource will create a new subscription using the information supplied in the *subscriptionRequest* element contained in the POST body, subject to access policy. A successful operation will return the new subscription.

Once a subscription has been successfully created on the server, the server will immediately send notifications for all documents matching the filter criteria independent of the event filter.

Header Parameters

The following header parameters are supported for the request for a new subscription resource.

Parameter	Value	Description
Content-Type	String	Identifies the content type encoding of the POST body

		contents. Must be a content type supported by the protocol.
Accept	String	Identifies the content type encoding requested for the returned results. Must be a content type supported by the protocol.

Body Parameters

The POST request must contain the *subscriptionRequest* element containing the initial parameters of the *subscription* resource to be created.

Parameter	Value	Description
requesterId	xsd:string	The identifier the requesting DDS requester would like to use for unique identification. An NSA must use its unique NSA identifier for *requesterId*.
callback	xsd:anyURI	The HTTP endpoint on the DDS requester host that will receive the notifications delivered for this subscription.
filter	FilterType	The *filter* criteria to apply to document events to determine if a notification should be sent to the DDS requester.

Returns

The following information can be returned in response to the POST.

Status Code	Element	Description
201	*subscription*	Returns a copy of the new subscription resource created as the result of a successful operation. The HTTP *Location* header field will contain the URI of the new subscription resource.
400	*error*	Returned if a DDS requester specifies an invalid request. An *error* element will be included populated with appropriate error information.
401	*error*	Returned if the DDS requester is not authorized to perform the requested operation or access the targeted resource. An *error* element will be included populated with appropriate error information.
500	*error*	Returned if an internal server error occurred during the processing of this request. An *error* element will be included populated with appropriate error information.

Example

The following example shows a valid ***POST*** request on the "*/subscriptions*" resource:

```
POST /subscriptions HTTP/1.1
Accept: application/vnd.ogf.nsi.dds.v1+xml
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:subscriptionRequest
        xmlns:tns="http://schemas.ogf.org/nsi/2013/04/discovery/types"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <requesterId>urn:ogf:network:example.com:2013:nsa:dasher</requesterId>
    <callback>http://dasher.example.com/discovery/callback</callback>
    <filter>
```

```
        <include>
            <event>All</event>
        </include>
    </filter>
</tns:subscriptionRequest>

HTTP/1.1 201 Created
Date: Mon, 10 Feb 2014 22:12:59 GMT
Content-Length: 405
Last-Modified: Mon, 10 Feb 2014 22:12:05 GMT
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
Location: /subscriptions/9e223d413578
<?xml version="1.0" encoding="UTF-8"?>
<tns:subscription
        id="9e223d413578"
        href="/subscriptions/9e223d413578"
        version="2014-02-10T22:12:05Z">
    <requesterId>urn:ogf:network:example.com:2013:nsa:dasher</requesterId>
    <callback>http://dasher.example.com/discovery/callback</callback>
    <filter>
        <include>
            <event>All</event>
        </include>
    </filter>
</tns:subscription>
```

11.2.8 getSubscription

Method: GET /subscriptions/{id}

Returns a *subscription* element containing the subscription instance identified by the *{id}* parameter of the subscription, subject to access policy.

Header Parameters

The following header parameters are supported for the subscriptions resource.

Parameter	Value	Description
Accept	String	Identifies the content type encoding requested for the returned results. Must be a content type supported by the protocol.
If-Modified-Since	RFC1123 date string	Constrains the GET request to return the matching subscription only if it has been updated since the time specified in this parameter. If the subscription resource does not meet these criteria, a 304 (not modified) response will be returned without any message-body.

Query Parameters

None.

Returns

The following information can be returned in response to the GET of a subscription.

Status Code	Element	Description
200	*subscription*	Successful operation returns the subscription identified by *id* in a *subscription* element. The *Last-Modified* header parameter will contain the time this

		subscription resource was last modified.
304	NA	Successful operation where there were no changes to the subscription resource identified by *id* given the *If-Modified-Since* criteria. Returns no message body.
400	*error*	Returned if a DDS requester specifies an invalid request. An *error* element will be included populated with appropriate error information.
401	*error*	Returned if the DDS requester is not authorized to perform the requested operation or access the targeted resource. An *error* element will be included populated with appropriate error information.
404	*error*	Returned if the requested subscription was not found. An *error* element will be included populated with appropriate error information.
500	*error*	Returned if an internal server error occurred during the processing of this request. An *error* element will be included populated with appropriate error information.

Example

The following example shows a valid ***GET*** request on the resource identified by *id="9e223d413578",* and URI "*/subscriptions/9e223d413578*". The result is a single *subscription* resource matching the specified *id*:

```
GET /subscriptions/9e223d413578 HTTP/1.1
Accept: application/vnd.ogf.nsi.dds.v1+xml

HTTP/1.1 200 OK
Date: Mon, 10 Feb 2014 22:12:59 GMT
Content-Length: 405
Last-Modified: Mon, 10 Feb 2014 22:12:05 GMT
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:subscription
        id="9e223d413578"
        href="/subscriptions/9e223d413578"
        version="2014-02-10T22:12:05Z">
    <requesterId>urn:ogf:network:example.com:2013:nsa:dasher</requesterId>
    <callback>http://dasher.example.com/discovery/callback</callback>
    <filter>
        <include>
            <event>All</event>
        </include>
    </filter>
</tns:subscription>
```

11.2.9 editSubscription

Method: PUT /subscriptions/{id}

The PUT operation on the *"/subscriptions/{id}*" resource will allow a DDS requester to edit the subscription corresponding to the identifier *{id},* subject to access policy, using the information supplied in the *subscriptionRequest* element contained in the PUT body. A successful operation will return the modified subscription.

Header Parameters

The following header parameters are supported for the update request for a subscription resource.

Parameter	Value	Description
Content-Type	String	Identifies the content type encoding of the PUT body contents. Must be a content type supported by the protocol.
Accept	String	Identifies the content type encoding requested for the returned results. Must be a content type supported by the protocol.

Body Parameters

The PUT request must contain the *subscriptionRequest* element containing the existing parameters of the *subscription* resource if they were not modified, as well as any new/edited values. For example, if the filter parameter is being edited, then the *requesterId* and *callback* URI must be supplied with their existing values.

Parameter	Value	Description
requesterId	xsd:string	The identifier the requesting DDS requester would like to use for unique identification. An NSA must use its unique NSA identifier for *requesterId*.
callback	xsd:anyURI	The HTTP endpoint on the DDS requester that will receive the notifications delivered for this subscription.
filter	FilterType	The *filter* criteria to apply to document events to determine if a notification should be sent to the DDS requester.

Returns

The following information can be returned in response to the PUT.

Status Code	Element	Description
200	*subscription*	Returns a copy of the modified subscription resource as the result of a successful operation.
400	*error*	Returned if a DDS requester specifies an invalid request. An *error* element will be included populated with appropriate error information.
401	*error*	Returned if the DDS requester is not authorized to perform the requested operation or access the targeted resource. An *error* element will be included populated with appropriate error information.
404	*error*	Returned if the requested subscription was not found. An *error* element will be included populated with appropriate error information.
500	*error*	Returned if an internal server error occurred during the processing of this request. An *error* element will be included populated with appropriate error information.

Example

The following example shows a valid ***PUT*** request on the "*/subscription/9e223d413578*" resource, editing the *filter* to include a new Updated event for the NSA "dasher". Notice that only those parameters that can be edited are included. In addition, the updated subscription resource will have a new version number corresponding to this update.

```
PUT /subscriptions/9e223d413578 HTTP/1.1
```

```
Accept: application/vnd.ogf.nsi.dds.v1+xml
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:subscriptionRequest
        xmlns:tns="http://schemas.ogf.org/nsi/2013/04/discovery/types"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <requesterId>urn:ogf:network:example.com:2013:nsa:dasher</requesterId>
    <callback>http://dasher.example.com/discovery/callback</callback>
    <filter>
        <include>
            <event>New</event>
        </include>
        <include>
            <event>Updated</event>
            <or><nsa>urn:ogf:network:example.com:2013:nsa:prancer</nsa></or>
        </include>
    </filter>
</tns:subscriptionRequest>

HTTP/1.1 200 OK
Date: Mon, 10 Feb 2014 22:20:59 GMT
Content-Length: 556
Last-Modified: Mon, 10 Feb 2014 22:20:58 GMT
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:subscription
        id="9e223d413578"
        href="/subscriptions/9e223d413578"
        version="2014-02-10T22:20:58Z">
    <requesterId>urn:ogf:network:example.com:2013:nsa:dasher</requesterId>
    <callback>http://dasher.example.com/discovery/callback</callback>
    <filter>
        <include>
            <event>All</event>
        </include>
        <include>
            <event>Updated</event>
            <or><nsa>urn:ogf:network:example.com:2013:nsa:prancer</nsa></or>
        </include>
    </filter>
</tns:subscription>
```

11.2.10 deleteSubscription

Method: DELETE /subscriptions/{id}

Deletes the *subscription* resource identified by the *{id} URI* parameter if access control permissions allow the DDS requester to perform the delete operation on the target resource.

Header Parameters

None.

Query Parameters

None.

Returns

The following information can be returned in response to the DELETE of a subscription.

Status Code	Element	Description
204	*NA*	Successful delete operation returns no content.
400	*error*	Returned if a DDS requester specifies an invalid request. An *error* element will be included populated with appropriate

		error information.
401	*error*	Returned if the DDS requester is not authorized to perform the requested operation or access the targeted resource. An *error* element will be included populated with appropriate error information.
404	*error*	Returned if the requested subscription was not found. An *error* element will be included populated with appropriate error information.
500	*error*	Returned if an internal server error occurred during the processing of this request. An *error* element will be included populated with appropriate error information.

Example

The following example shows a valid ***DELETE*** request on the resource identified by *id="9e223d413578",* and URI "*/subscriptions/9e223d413578*". The result is a single *subscription* resource matching the specified *id*:

```
DELETE /subscriptions/9e223d413578 HTTP/1.1

HTTP/1.1 204 No Content
Date: Mon, 10 Feb 2014 22:12:59 GMT
```

11.2.11 Notifications

When a document event occurs matching a registered subscription the DDS provider must issue a *notification* to the DDS requester endpoint identified in the *subscription* resource. Multiple events can be grouped and delivered together in a single notification if these events occur within a reasonable period of time of each other. Notification delivery should not be delayed.

Notifications are also sent when a subscription is first created and will include any documents matching the initial filter criteria.

A failure in notification delivery SHOULD result in the deletion of the subscription. Retries are possible but are not required. Notifications should not be discarded without deleting the subscription. To detect delivery failures the DDS requester MUST periodically verify that subscriptions are still valid on the DDS provider.

By creating a subscription, the DDS requester has entered a contractual agreement to expose an HTTP endpoint capable of receiving a POST operation with a message body containing a *notifications* element using the content encoding of the original subscription.

Method: POST <DDS requester supplied endpoint>

The POST operation on the *"<DDS requester supplied endpoint>*" is a remote call from the DDS provider holding the subscription to the DDS requester endpoint registered in the subscription. The DDS requester must return an HTTP 202 status code in response to the POST indicating it has successfully accepted the notification. Any other return code results in a deletion of the subscription.

A server may periodically issue a POST to the DDS requester endpoint with a notification element containing zero elements. This should not be considered an error and the DDS requester MUST return an HTTP 202 status code in response. The server to check the validity of a subscription can use this.

Header Parameters

The following header parameters are supported for the notification request to the DDS requester endpoint.

Parameter	Value	Description
Content-Type	String	Identifies the content type encoding of the POST body contents. Must be identical to the value as used by the DDS requester on subscription.

Body Parameters

The POST request must contain the *notifications* element, which will contain the list of zero or more notifications matching the subscription filter.

Parameter	Value	Description
providerId	xsd:anyURI	The identifier of the DDS provider generating the notification. This is the provider on which the subscription was created.
id	xsd:string	The identifier of the subscription that generated the notifications.
href	xsd:anyURI	The URI reference for subscription that generated the notification. This can be used to directly access the subscription.
discovered	xsd:dateTime	The most recent document discovery time for the server in the context of when the notification was generated.
notification	NotificationListType	A list of zero or more notifications matching the subscription filter criteria.

Returns

The DDS requester receiving the notification must return an HTTP 202 status code in response to the POST. Any other status code will result in a deletion of the subscription.

Status Code	Element	Description
202	*NA*	Indicates the subscribed DDS requester has accepted the notification for processing.

Example

The following example shows a notification ***POST*** request on the "*/requesterEndpoint*" resource:

```
POST /requesterEndpoint HTTP/1.1
Content-Type: application/vnd.ogf.nsi.dds.v1+xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:notifications xmlns:tns="http://schemas.ogf.org/nsi/2013/04/discovery/types"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       providerId="urn:ogf:network:example.com:2013:nsa:vixen"
       id="9e223d413578"
       href="/subscriptions/9e223d413578">
    <discovered>2014-02-10T22:20:58Z</discovered>
    <tns:notification>
        <discovered>2014-02-10T22:20:58Z</discovered>
        <event>New</event>
        <document id="urn:ogf:network:example.com:2013:network:lincolntunnel"
            version="2014-02-10T22:15:10Z" expires="2014-02-11T22:15:10Z">
            <nsa>urn:ogf:network:example.com:2013:nsa:prancer</nsa>
            <type>application/vnd.ogf.nsi.topology.v2+xml</type>
            <signature> ... </signature>
            <content> ... </content>
        </document>
```

```
    </tns:notification>
</tns:notifications>

HTTP/1.1 202 Accepted
Date: Mon, 10 Feb 2014 22:12:59 GMT
Content-Length: 0
```

12 Security Considerations

Documents carried by the NSI Document Distribution Service must be verifiable by DDS requesters and DDS providers within the GDS (e.g. the requester agent must be able to determine that the content of the document was not altered during delivery, and is in fact, the same document published by the source provider). The NSI Document Distribution Service includes an element in the document meta-data to allow for the association of a digital signature by the publishing NSA, which can then be used by each requester within the GDS to validate the authenticity of the attached document. The type of digital signature and algorithms used is left for definition outside of this specification since it may be document specific.

It is also assumed that exchange of documents between the DDS requester and provider roles is secured to the level of other protocols within the NSI protocol suite. This security must include authentication, authorization, and confidentiality. To maintain consistency with other NSI specifications, the following security strategy is incorporated from [OGF NSI-CS].

A DDS deployment MUST use TLS to ensure secure communication between DDS requester and DDS provider entities. TLS provides message integrity, confidentiality and authentication via the X.509 certificates, and protects against replay attacks. TLS version 1.2 MUST be supported.

Trust between DDS servers is pairwise and MUST be established out-of-band.

Authorization is done above the TLS transport at the DDS application level. Protection MUST be provided against unauthorized entities changing document entries within the DDS. A DDS server SHOULD use the X.509 certificate DN of the requesting DDS entry as the identifying attribute for authorization.

To address the individual DDS API security requirements the following implementation is recommended:

1. *Notifications MUST only be accepted from trusted "peer" DDS providers for which valid subscriptions have been created. Unsolicited notification MUST be discarded.*

   A DDS requester creating remote subscriptions maintains a list consisting of the following pairs [remote *subscriptionId*, provider's X.509 certificate DN]. When a notification arrives from a peer DDS server the X.509 certificate DN on the incoming notification TLS connection and the *subscriptionId* from the incoming notification message is compared against the stored provider DN/*subscriptionId* pair. If they match then the notification is accepted, if they do not match then the notification is rejected.

2. *Addition of new documents and updates to existing documents within a DDS provider MUST be restricted to authorized DDS requesters.*

   A DDS provider maintains a list of X.509 certificates DN for DDS requesters allowed to created or modify documents stored within the GDS. Additional access control policies will need to be defined identifying:

   - The allowable value for the owning NSA element.

- The types of documents a DDS requester may create or modify.

3. *Read access (get operations) SHOULD be restricted to only authorized DDS requesters.*

   A DDS provider maintains a list of X.509 certificates DN for DDS requesters allowed read access to the GDS. Additional read granularity based on the DDS requester DN MAY be provided if required.

4. *Creation of subscription-based notifications SHOULD be restricted to authorized DDS requesters.*

   A DDS provider maintains a list of X.509 certificates DN for DDS requesters allowed to subscribe for notifications on documents stored within the GDS. Additional subscription granularity (document type, instance, etc.) based on the DDS requester DN may be provided if required.

5. *Editing and deletion of subscriptions SHOULD be restricted to the DDS requester associated with the subscription.*

   A DDS provider creating local subscriptions maintains a list consisting of the following pairs [*subscriptionId*, requester's X.509 certificate DN]. When a DDS requester attempts to modify or delete a subscription, the DDS provider compares the X.509 certificate DN on the incoming client TLS connection and the *subscriptionId* from the incoming request message against the stored requester DN/subscriptionId pair. If they match then the requested operation is accepted, if they do not match then the operation is rejected. Additional operation granularity based on the DDS requester DN may be provided if required.

13 Glossary

Aggregator NSA (AG)	The Aggregator NSA is a Provider Agent that acts as both a requester and provider NSA. It can service requests from other NSA, perform path finding, and distribute segment requests to child NSA for processing.
Connection Service (CS)	The NSI Connection Service is a service that allows an RA to request and manage a Connection from a PA. See [OGF NSI-CS].
Document Distribution Service (DDS)	The NSI Document Distribution Service is a RESTful web service allows the exchange of documents between the DDS requester and provider agent participating in a Global Document Space. The NSA Description Document is an example of information exchanged using the DDS.
DDS Requester	The client that request documents from the DDS provider
DDS Provider	The server that provides DDS documents to the DDS requester
Global Document Space (GDS)	A logical space that consists of all documents published by the set of interconnected DDS providers implementing the DDS.
Network Service Agent (NSA)	The Network Service Agent is a concrete piece of software that sends and receives NSI Messages. The NSA includes a set of capabilities that allow Network Services to be delivered.
Network Service Interface (NSI)	The NSI is the interface between RAs and PAs. The NSI defines a set of interactions or transactions between these NSAs to realize a Network Service.
Network Services Framework (NSF)	The Network Services framework describes an NSI message-based platform capable of supporting a suite of Network Services such as

	the Connection Service and the Topology Service. See [OGF NSF].
NSA Description document	The NSA Description document encapsulates descriptive meta-data associated with an NSA such as all NSI services and associated protocol interfaces offered by the NSA.
NSI Topology	The NSI Topology defines a standard ontology and a schema to describe network resources that are managed to create the NSI service. The NSI Topology as used by the NSI CS (and in future other NSI services) is described in [OGF NSI-TOP].
Requester/Provider Agent (RA/PA)	An NSA acts in one of two possible roles relative to a particular instance of an NSI. When an NSA requests a service, it is called a Requester Agent (RA). When an NSA realizes a service, it is called a Provider Agent (PA). A particular NSA may act in different roles at different interfaces.
NSI Service Definition	A document describing the service offered by an NSA and it's underlying Network. A Network can offer multiple services, and therefore, have multiple Service Definitions defined.
Service Plane	The collection of network resources over which the service is delivered.
Simple Object Access Protocol (SOAP)	SOAP is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks.
Ultimate PA (uPA)	The ultimate PA is a Provider Agent that has an associated NRM.
Ultimate RA (uRA)	The Ultimate RA is a Requester Agent is the originator of a service request.
XML Schema Definition (XSD)	XSD is a schema language for XML. See [W3C XSD]
eXtensible Markup Language (XML)	XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

14 Contributors

John H. MacAuley, ESnet, macauley@es.net

15 Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights, which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

16 Disclaimer

This document and the information contained herein is provided on an "As Is" basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

17 Full Copyright Notice

18 References

[RFC 2119]. Scott Bradner. Key Words for Use in RFCs to Indicate Requirement Levels, RFC 2119. The Internet Society. March 1997. http://tools.ietf.org/html/rfc2026

[RFC 6350] Simon Perreault. vCard Format Specification RFC 6350 (Standards Track), August 2011. URL http://tools.ietf.org/html/rfc6350.

[RFC 6351] S. Perreault. xCard: vCard XML Representation RFC 6351 (Standards Track), August 2011. URL http://tools.ietf.org/html/rfc6351.

[GFD.213] Guy Roberts, et al. "OGF Network Service Framework v2.0", Group Working Draft (GWD), candidate Recommendation Proposed (R-P), January 28, 2014.

[GFD.212] Guy Roberts, et al. "OGF NSI Connection Service v2.0", Group Working Draft (GWD), candidate Recommendation Proposed (R-P), January 12, 2014.

[OGF NSI-ND] John MacAuley, et al. "Network Service Interface NSA Description Document v1.0", Group Working Draft (GWD), candidate Recommendation Proposed (R-P), June 3, 2015.

[OGF NSI-NSIPF] John MacAuley, et al. "GFD-I.217 NSI Signaling and pathfinding", Grid Forum Document Informational, May 1, 2015

[OGF NML] OGF GFD.206: Network Markup Language Base Schema version 1, http://www.gridforum.org/documents/GFD.206.pdf

[W3C XSD] W3C XML "Schema Definition Language (XSD) 1.1 Part 2: Datatypes", http://www.w3.org/TR/xmlschema11-2/#anyURI

[FIELDING] R. T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. UNIVERSITY OF CALIFORNIA, IRVINE, 2000, Chapter 5.

[RICH] L. Richardson, et al. Restful Web Services. O'Reilly Media; First Edition, May 15, 2007.

19 Appendix I –Topology distribution requirements

This appendix is informational only.

The key motivation for the development of the NSI DDS is to be able share NSI topology documents. The following requirements were identified.

- The solution must allow NSI topology information to be shared between NSAs.
- The solution must allow AG NSAs to aggregate topology.
- The solution must support chain based path signaling.
- The solution must support tree based path signaling.
- The solution must support centralized path finding for source-based routing decisions.
- The solution must support distributed path finding for hop-by-hop routing decisions.
- NSA description document must include <peersWith> and <feature> elements are used to build a directed control plane graph for message routing.
- NSA description document must include nsaId to networkId mappings to determine which NSA gets messages for a specific network.
- NSA description document must include interface elements for protocol endpoints.
- The solution must allow the creation of a full view of network topology to perform advanced "intelligent" routing decisions.
- Service description documents for all networks must be able to determine the constraints and parameters of the services offered.
- The solution must be able to support application/project/deployment specific aggregators for use by specialized user groups.
- The solution must be able to deploy core aggregators that perform path finding but are user agnostic. These aggregators will not know the identity of the user, nor the end user authentication schemes (uPA specific).
- In most cases the uRA associated with the end user will have no concept of path finding or network topology, so must be able to delegate the path finding function to an aggregator within the network.

20 Appendix II – Document payload sizes and rate of change

This appendix is informational only.

Document Payload Sizes
With any flooding-based protocol it is important to understand both the behavior and volume of data to be exchanged by the protocol. By building these data models it is possible to determine the operational parameters of the protocol, and understand the limiting factors. In the case of the NSI Document Distribution Service there are two documents currently defined that will need to be supported by the protocol. These documents and associated sizes are shown below.

Document	Uncompressed	Compressed
NSA Description Document	5 KB	2 KB
NSI Topology (1,000 ports)	1.5 MB	85 KB

NSI Topology (300 ports)	450 KB	26 KB

Table 4 – Physical document sizes

The NSA Description Document [OGF NSI-ND] is a relatively small XML document with an estimated upper limit of 5 Kbytes in size, and a compressed size of 2 Kbytes. The larger of the two documents is the NML Topology Document [OGF NSI-NML], which is directly dependent on the number of logical ports being modeled within a Network. In Table 4, a fully specified NSI Topology Document was defined using the XML representation for a Network of 1,000 bidirectional ports using PortGroup summarization. This reference model assumed 30% E-NNI (inter-domain) and 70% UNI (client) ports. When all 1,000 ports were modeled it resulted in an uncompressed document size of 1.5 Mbytes and a compressed size of 85 Kbytes. If only the E-NNI ports were modeled for path computation, then the document size was reduced to 450 Kbytes uncompressed and 26 Kbytes compressed. Reducing the information model will have impact on advanced path finding (i.e. adaptation) and is open for further study.

To further reduce document sizes an alternative representation such as JSON could be used to remove the verbosity of the current XML definitions.

It should be noted that NSI Topology Documents represent the bulk of document data held within the GDS. The volume of this data is directly related to the number of Networks advertised by uPAs, and the number of ports publically visible within these networks. Aggregator NSAs only generate NSA Description Documents, while RA generate no documents.

Global network size	**Combined sizes (uncompressed)**	**Combined sizes (compressed)**
10,000 networks	14.6 GB	850 MB
5,000 networks	7.3 GB	425 MB
1,000 networks	1.5 GB	85 MB
500 networks	750 MB	42 MB

Table 5 – Combined document sizes for average network size of 1,000 ports

Table 5 shows the combined document sizes for interconnected Network sizes ranging from 500 Networks through 10,000 Networks each advertising 1,000 ports within their NSI Topology Documents. Numbers are provided for both uncompressed and compressed document content.

Global network size	**Combined sizes (uncompressed)**	**Combined sizes (compressed)**
10,000 networks	4.3 GB	273 MB
5,000 networks	2.2 GB	137 MB
1,000 networks	444 MB	27 MB
500 networks	222 MB	14 MB

Table 6 – Combined document sizes for average network size of 300 ports

In Table 6 we see similar numbers but with each Network only reporting 300 ports within their NSI Topology Documents. These numbers would represent the advertising of only the inter-network E-NNI ports.

Document rate of change

The DDS protocol does not dictate a specific period to update or refresh a document. This behavior is dependent on the type of data being modeled within the document published to the GDS. When a new version of a document is available, it is published into the GDS using a new version. An NSA can also re-publish an existing document into the DDS if it would like to refresh the current version of the document. If the version of the document is already present, the re-

published version will be ignored. If however if it is not, it will be added to the GDS following the defined document versioning rules.

The DDS protocol is agnostic to document content and has no facility to provide a mechanism for incremental document updates. This is left for future work.

There is an expectation that larger documents distributed by the DDS protocol will be relatively static in nature requiring infrequent updates. The more frequent a document requires updating, the more impact it has on bandwidth consumed for flooding between providers. Taking the maximum (850 MB) and minimum (42 MB) values from Table 5 we can see a large gap in the bandwidth requirements if all documents within the GDS were updated once a day.

- 850 MB over 24-hour period is an average 81 Kb/s * # of peers.
- 42 MB over 24-hour period is an average 4 Kb/s * # of peers.

Based on the relatively static nature of the NSA Description and the NSI Topology documents we can expect updates less frequently that once a day. As new document types are defined and propagated through the DDS care will need to be given to avoid excessive strain on resources.

21 Appendix III – DDS provider Pseudo Code

The following appendix contains example pseudo code for the DDS provider function. The pseudo code describes the DDS abstract API logic, and can be used to implement the DDS function within an NSI deployment.

The NSI CS Aggregator NSA will deploy a full DDS provider performing both requester and provider functions. The Aggregator NSA registers for document notification from all peer NSA, and delivers document notifications to all subscribed peers. The Aggregator also publishes documents associated with its own NSA such as an NSA description document. An Aggregator would use the addDocument/updateDocument API or some locally defined mechanism to publish these documents into the local DDS provider instance, thereby allowing them to be propagated to all peers forming the GDS.

The NSI CS uPA NSA does not require access to documents published by other NSA within the GDS. For this reason, the uPA has two implementation options for integration into the DDS. The first is to use a DDS requester to publish its documents (addDocument/updateDocument API) into an Aggregator that will maintain the lifecycle of the documents on behalf of the uPA. This will require a prearranged agreement between the uPA and Aggregator.

The second option is for the uPA to deploy a DDS provider but only enable the provider role. In this configuration the DDS provider allows peer Aggregators to subscribe for notifications on document events relating to the uPA's documents, but does not itself subscribe to any peer NSA for document notifications. This will result in only the uPA's documents being contained in the local DDS provider, with all peer NSA being updated with uPA document notifications.

```
PROGRAM DdsServer:

    // Global variables holding configuration, state, and discovered documents.
    DECLARE a list variable called Peers holding configuration information for all peers;
    DECLARE a map variable called GlobalDocumentSpace holding all known documents in the
            GDS(indexed by unique document identifier);
    DECLARE a map variable called LastDiscovered holding discovered date/time values for
            each document (indexed by unique document identifier);
```

```
DECLARE a map variable called MySubscriptions holding local subscriptions on remote
        DDS providers (indexed by peer containing subscription);
DECLARE a map variable called PeerSubscriptions holding remote DDS provider
        subscriptions on local DDS provider(indexed by peer owning subscription);
DECLARE a string variable called MyNsaId holding the local NSA identifier;
DECLARE a time variable called SubscriptionAuditInterval holding the time between
        subscription audit intervals;
DECLARE a time variable called ExpireAuditInterval holding the time between document
        expiry audit intervals;

// start() initializes the system and registers subscriptions with all remote DDS
// server Peers.
PROCEDURE start() {
    // Initialize the DDS system.
    READ Peers from list of peer NSA from configuration;
    READ SubscriptionAuditInterval from configuration;
    READ ExpireAuditInterval from configuration;
    READ MyNsaId from configuration;
    READ GlobalDocumentSpace from storage discarding any expired documents;

    SET MySubscriptions to an empty map<peer, subscription>;
    SET PeerSubscriptions to an empty map<peer, subscription>;

    // For simplification register for all document events on all Peers configured as
    // a provider role.  Each peer will send a full list of documents present in their
    // document space.
    FOR each peer in Peers with a provider role DO
        // First we need to delete any existing subscriptions we may have on this
        // peer.
        CALL peer.getSubscriptions(MyNsaId)
                  RETURNING status, subscriptions, and lastModifiedTime;
        IF status is success THEN
            FOR each subscription in subscriptions DO
                CALL peer.deleteSubscription(subscription.id);
            ENDFOR;
        ENDIF;

        // Add the new subscription and store it for later auditing.
        CALL peer.addSubscription(MyNsaId, notificationCallback,
                filter(include event All)) RETURNING status, subscription, and
                lastModifiedTime;
        IF status is success and subscription is present THEN
            STORE <peer, subscription> in MySubscriptions;
        ENDIF;
    ENDFOR;

    // Schedule maintenance tasks.
    SCHEDULE subscriptionAudit() at SubscriptionAuditInterval;
    SCHEDULE documentExpireAudit() at ExpireAuditInterval;
}


// subscriptionAudit() verifies there is an active subscription on all configured DDS
// Peers. It will create a new subscription if one does not exist, and will delete any
// subscriptions no longer in use.
PROCEDURE subscriptionAudit() {
    // oldSubscriptions will hold the list of MySubscriptions we need to clean up when
    // audit is completed.
    DECLARE a map variable called oldSubscriptions to hold the list of MySubscriptions
            to clean up when audit is completed (indexed by peer containing the
            subscription);
    SET oldSubscriptions to copy of MySubscriptions;

    // Audit subscription for each of our configured Peers.
    FOR each peer in Peers with a provider role DO
        SET subscription to MySubscriptions.get(peer);

        IF subscription is present THEN
            // Get subscription for this peer.
```

```
            CALL peer.getSubscription(subscription.id) RETURNING oldSubscription;

            // Remove this subscription from our cleanup list.
            REMOVE oldSubscription from oldSubscriptions;

            IF oldSubscription is present THEN
                // This subscription is still valid so proceed to next iteration.
                CONTINUE;
            ENDIF;

            // This subscription is no longer valid.
            REMOVE subscription from MySubscriptions;

        ENDIF;

        // We do not have a subscription for this peer so create one.
        CALL peer.addSubscription(MyNsaId, notificationCallback,
                filter(include event All)) RETURNING newSubscription;

        IF newSubscription is present THEN
            STORE <peer, newSubscription> in MySubscriptions;
        ENDIF;
    ENDFOR;

    // Now remove any MySubscriptions no longer needed.
    FOR each subscription in oldSubscriptions DO
        SET peer to subscription.peer;
        CALL peer.deleteSubscription(subscriptionId);
    ENDFOR;

    // Schedule our next audit run.
    SCHEDULE subscriptionAudit() at SubscriptionAuditInterval;
}

// documentExpireAudit() - removes any expired documents from the local document
// space.
PROCEDURE documentExpireAudit() {
    FOR each document in GlobalDocumentSpace DO
        IF document.expires is in past THEN
            REMOVE document from GlobalDocumentSpace;
        ENDIF;
    ENDFOR;

    // Schedule our next audit run.
    SCHEDULE documentExpireAudit() at ExpireAuditInterval;
}

// notificationCallback() is the notification callback endpoint for delivery of
// subscription events from remote DDS Peers.
API notificationCallback(notifications) RETURNS status {
    VALIDATE parameters notifications RETURNING failed if invalid;

    // Reject the notification if not from a valid peer.
    IF notifications.providerId not in list of Peers with a provider role THEN
        RETURN status of failed(invalid peer);
    ENDIF;

    // Reject the notification if not a valid subscription.
    IF notifications.id not in list of MySubscriptions THEN
        RETURN status of failed(invalid subscription);
    ENDIF;


    // Process each notification, storing new/updated documents and propagating any
    // changes to peers.
    FOR each notification in notifications DO
        // Get document out of notification.
        SET document to notification.document;
```

```
            // Create a unique document identifier for indexing.
            CALL uid(document.nsa, document.type, document.id) RETURNING uid;

            // If an old version of the document is present make sure this is a newer
            // version before storing and propagating.
            SET oldDocument to GlobalDocumentSpace.get(uid);
            IF oldDocument is present THEN
                IF oldDocument.version is less than document.version THEN
                    REPLACE oldDocument in GlobalDocumentSpace with document;
                    STORE current date/time in LastDiscovered indexed by uid;
                    CALL propagateDocument(providerId, UPDATE, document);
                ENDIF;
            ELSE
                STORE document in GlobalDocumentSpace indexed by uid;
                STORE current date/time in LastDiscovered for uid;
                CALL propdateDocument(providerId, NEW, document);
            ENDIF;
        ENDFOR;
    }

    // propdateDocument() sends document notification events to all DDS peer subscribed
    // for the document event type.
    PROCEDURE propagateDocument(providerId, event, document) {
        // Inspect each subscription to see if it matches this document event.
        FOR each subscription in PeerSubscriptions DO
            // Do not send the document event back to the originating provider.
            IF subscription.requesterId equals providerId THEN
                CONTINUE;
            ENDIF;

            // If the subscription matches the document even propagate.
            IF subscription.filter matches event and document THEN
                SET callback to subscription.callback;
                SET notification to new notification(MyNsaId, event, document);
                CALL callback(notification) RETURNING status;

                // Subscription may no longer be valid.  Delete and let peer
                // re-register their next audit.
                IF status is not success THEN
                    DELETE subscription from PeerSubscriptions;
                ENDIF;
             ENDIF;
        ENDFOR;
    }


    // getDocuments() returns a list of documents and the time of the latest document
    // change on the DDS provider.
    API getDocuments([nsa], [type], [id], [lastDiscoveredTime])
            RETURNS status, a list of [0..n] document, and [lastDiscoveredTime] {
        VALIDATE parameters nsa, type, id, and lastDiscoveredTime
                RETURNING status of failed(invalid parameter) if invalid;

        DECLARE a list variable called results to hold documents matching the
                query filter;
        DECLARE a date/time variable called newLast to hold the time of the most recently
                discovered document;

        SET newLast to Date(0);

        IF lastDiscoveredTime is absent THEN
            SET lastDiscoveredTime to Date(0);
        ENDIF;

        // Inspect each document in the GDS for a match.
        FOR each document in GlobalDocumentSpace DO
            // Create a unique document identifier for indexing.
            CALL uid(document.nsa, document.type, document.id) RETURNING uid;
```

```
            // Determine if this document meets any lastDiscoveredTime criteria.
            DECLARE a date/time variable called currentLast to hold the current document's
                    last discovered time;
            SET currentLast to LastDiscovered.get(uid);
            IF currentLast is later than lastDiscoveredTime THEN
                // Now match on the other criteria.
                IF document matches filter(nsa, type, id) THEN
                    STORE document in results;

                    // Track the latest discovered time.
                    IF currentLast is later than newLast THEN
                        STORE currentLast in newLast;
                    ENDIF;
                ENDIF;
            ENDIF;
        ENDFOR;

        RETURN status of success, results, and newLast;
    }

    // getLocalDocuments() returns a list of documents associated with the queried DDS
    // provider and the time of the latest document change on that provider.
    API getLocalDocuments([type], [id], [lastDiscoveredTime])
            RETURNS status, a list of [0..n] document, and [lastDiscoveredTime] {
        CALL getDocuments(MyNsaId, type, id, lastDiscoveredTime)
                RETURNS results and newLast;
        RETURN results and newLast;
    }

    // getDocument() returns the requested document and the time of the latest change
    // on the document.
    API getDocument(nsa, type, id, [lastDiscoveredTime])
            RETURNS status, [document], and [lastDiscoveredTime] {
        CALL getDocuments(nsa, type, id, lastDiscoveredTime) RETURNS results and newLast;
        RETURN results and newLast;
    }

    // addDocument() adds a new document to the space associated with the DDS provider.
    API addDocument(nsa, type, id, version, expires, [signature], content)
            RETURNS status, [document], and [lastDiscoveredTime] {
        VALIDATE nsa, type, id, version, expires, signature, and content
                RETURNING status of failed(invalid parameter) if invalid;

        // Build the unique document identifier and determine if document already exists.
        CALL uid(document.nsa, document.type, document.id) RETURNING uid;
        SET document to GlobalDocumentSpace.get(uid);

       // A document can only be added when one does not already exist.
        IF document is present THEN
            RETURN status of failed(document exists);
        ENDIF;

        // Add the new document.
        SET document to
                new document(nsa, type, id, version, expires, signature, content);
        STORE document in GlobalDocumentSpace indexed by uid;

        // Update the lastDiscoveredTime.
        SET lastDiscoveredTime as current date/time;
        STORE lastDiscoveredTime in LastDiscovered indexed by uid;

        // Send the new document event to all peers.
        CALL propagateDocument(MyNsaId, NEW, document);

        RETURN status of success, document, and lastDiscoveredTime;
    }

    // updateDocument - updates an existing document within the space associated with the
    // DDS provider.
```

```
API updateDocument(nsa, type, id, version, expires, [signature], content)
        RETURNS status, [document], and [lastDiscoveredTime] {
    VALIDATE nsa, type, id, version, expires, signature, and content
            RETURNING status of failed(invalid parameter) if invalid;

    // Build the unique document identifier and retrieve the document for update.
    CALL uid(document.nsa, document.type, document.id) RETURNING uid;
    SET document to GlobalDocumentSpace.get(uid);

    // A document must be present to update.
    IF document is not present THEN
        RETURN status of failed(document does not exists);
    ENDIF;

    // Update only if this is a new document.
    IF document.version is not less than version THEN
        RETURN status of failed(invalid version);
    ENDIF;

    // Replace existing document with the updated document.
    SET updatedDocument to
            new document(nsa, type, id, version, expires, signature, content);
    REPLACE document in GlobalDocumentSpace with updatedDocument;

    // Update the lastDiscoveredTime.
    SET lastDiscoveredTime as current date/time;
    STORE lastDiscoveredTime in LastDiscovered indexed by uid;

    // Send document update event to all peers.
    CALL propagateDocument(MyNsaId, UPDATE, document);

    RETURN status of success, document, and lastDiscoveredTime;
}

// addSubscription() subscribes a requester for document event notifications based on
// the supplied filter.
API addSubscripton(requesterId, callback, filter)
            RETURNS status, [subscription], and [lastModifiedTime] {
    VALIDATE requesterId, callback, and filter
            RETURNING status of failed(invalid parameter) if invalid;

    // Verify this requesting peer is configured for a requester role.
    IF requesterId not in list of Peers with a requester role THEN
        RETURN status of failed(invalid peer);
    ENDIF;

    // Create the new subscription with a new unique subscription identifier.
    SET subscription to new subscription(requesterId, callback, filter);
    STORE subscription in PeerSubscriptions indexed by subscription.id;

    // Save the of this subscription's creation for lastModifiedTime queries.
    SET lastModifiedTime as current date/time;
    STORE lastModifiedTime in LastModified indexed by subscription.id;

    // Send a notification for all documents matching the new filter but with document
    // event All.
    FOR each document in GlobalDocumentSpace DO
        IF subscription.filter matches document THEN
            SET callback to subscription.callback;
            SET notification to new notification(MyNsaId, All, document);
            CALL callback(notification) RETURNING status;
            IF status is not success THEN
                DELETE subscription from PeerSubscriptions;
                RETURN status of failed(invalid endpoint);
            ENDIF;
        ENDIF;
    ENDFOR;

    RETURN status of success, subscription, and lastModifiedTime;
```

```
}

// editSubscription() allows an existing subscription to be edited.
API editSubscription(id, requesterId, callback, filter)
        RETURNS status, [subscription], and [lastModifiedTime] {
    VALIDATE id, requesterId, callback, and filter
            RETURNING status of failed(invalid parameter) if invalid;

    // Get the current subscription.
    SET subscription to PeerSubscriptions.get(id);

    // A subscription must be present to update.
    IF subscription is not present THEN
        RETURN status of failed(subscription does not exists);
    ENDIF;

    // Update the subscription.
    SET newSubscription to new subscription(requesterId, callback, filter);
    REPLACE subscription in PeerSubscriptions with newSubscription;

    // Updated the last modified time.
    SET lastModifiedTime as current date/time;
    STORE lastModifiedTime in LastModified indexed by subscription.id;

    // Build a list of notifications based on documents matching the updated filter
    // criteria.
    DECLARE a list variable called notifications to hold a list of notification for
            each document matching filter criteria;
    FOR each document in GlobalDocumentSpace DO
        IF newSubscription.filter matches document THEN
            SET notification to new notification(MyNsaId, All, document);
            STORE notification in notifications;
        ENDIF;
    ENDFOR;

    // Send list of notifications to the subscriber.
    SET callback to newSubscription.callback;
    CALL callback(notifications) RETURNING status;
    IF status is not success THEN
        DELETE newSubscription from PeerSubscriptions;
        RETURN status of failed(invalid endpoint);
    ENDIF;

    RETURN status of success, newSubscription, and lastModifiedTime;
}

// deleteSubscription() deletes the subscription associated with id from the provider
// NSA.
API deleteSubscription(id) RETURNS status, and [subscription] {
    VALIDATE id RETURNING status of failed(invalid parameter) if invalid;

    // Get the subscription.
    SET subscription to PeerSubscriptions.get(id);

    // A subscription must be present to delete.
    IF subscription is not present THEN
        RETURN status of failed(subscription not found);
    ENDIF;

    DELETE subscription from PeerSubscriptions;

    RETURN status of success and subscription;
}

// getSubscriptions() returns a list of subscriptions and the time of the latest
// subscription change on the provider NSA.
API getSubscriptions([requesterId], [lastModifiedTime])
        RETURNS status, list of [0..n] subscription, and [lastModifiedTime] {
    VALIDATE requesterId and lastModifiedTime
```

```
            RETURNING status of failed(invalid parameter) if invalid;

    DECLARE a list variable called results to hold the matching list of subscriptions;
    DECLARE a date/time variable called newLast to hold the most recent
            lastModifiedTime;

    SET newLast to Date(0);

    // If a lastModifiedTime filter was not provided set to start of time so all
    // subscriptions are more recent.
    IF lastModifiedTime is absent THEN
        SET lastModifiedTime to Date(0);
    ENDIF;

   // Add subscriptions that match the requested filter.
   FOR each subscription in PeerSubscriptions DO
        DECLARE a date/time variable called currentLast to hold this subscription's
                lastModifiedTime;
        SET currentLast to LastModified.get(subscription.id);
        IF currentLast is later than lastModifiedTime THEN
            IF subscription matches filter(requesterId, lastModifiedTime) THEN
                STORE subscription in results;

                IF currentLast is later than newLast THEN
                    STORE currentLast in newLast;
                ENDIF;
            ENDIF;
         ENDIF;
    ENDFOR;

    RETURN status of success, results, and newLast;
}

// getSubscription() returns a single subscription identified by the id parameter and
// the time this subscription was last modified.
API getSubscription(id, [lastModifiedTime])
        RETURNS status, [subscription], and [lastModifiedTime] {
    VALIDATE id and lastModifiedTime
            RETURNING status of failed(invalid parameter) if invalid;

    // Get the subscription.
    SET subscription to PeerSubscriptions.get(id);

    // A subscription must be present for this to be successful.
    IF subscription is not present THEN
        RETURN status of failed(subscription not found);
    ENDIF;

    DECLARE a date/time variable called currentLast to hold this subscription's
            lastModifiedTime;
    SET currentLast to LastModified.get(subscription.id);

    // If a lastModifiedTime filter was not provided set to start of time so all
    // subscriptions are more recent.
    IF lastModifiedTime is absent THEN
        SET lastModifiedTime to Date(0);
    ENDIF;

    IF currentLast is later than lastModifiedTime THEN
        RETURN status of success and subscription;
    ELSE
        RETURN status of success(not modified);
    ENDIF;
}

// getAll() returns a collection of subscriptions, documents, and local documents
// discovered since lastDiscoveredTime (treating lastDiscoveredTime as
// lastModifiedTime in the case of subscriptions). The time of the last
// discovered/modified element is also returned.
```

```
    API getAll([lastDiscoveredTime])
            RETURNS status, list of [0..n] subscription, list of [0..n] document,
            list of [0..n] local document, and [lastDiscoveredTime] {
        VALIDATE lastDiscoveredTime
                RETURNING status of failed(invalid parameter) if invalid;

        DECLARE a list variable called subscriptions to hold the matching list of
                subscriptions;
        DECLARE a list variable called documents to hold the matching list of documents;
        DECLARE a list variable called local to hold the matching list of local documents;
        DECLARE a variable called status to hold the return status of method calls;
        DECLARE a date/time variable called recentTime to hold the lastDiscoveredTime;
        DECLARE a date/time variable called currentLast to hold the individual call
                results;

        CALL getSubscriptions(NULL, lastModifiedTime)
                RETURNING status, subscriptions, and recentTime;
        IF status is failed THEN
            RETURN status;
        ENDIF;

        CALL getDocuments(NULL, NULL, NULL, lastDiscoveredTime)
                RETURNING status, documents, and currentLast;
        IF status is failed THEN
            RETURN status;
        ENDIF;

        IF currentLast is later than recentTime THEN
            SET recentTime to currentLast;
        ENDIF;

        CALL getLocalDocuments(NULL, NULL, lastDiscoveredTime)
                RETURNING status, local, and lastDiscoveredTime;
        IF status is failed THEN
            RETURN status;
        ENDIF;

        IF currentLast is later than recentTime THEN
            SET recentTime to currentLast;
        ENDIF;

       RETURN status of success, subscriptions, documents, local, and recentTime;
    }

END;
```

22 Appendix IV – NSI Document Distribution Service Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
The OGF takes no position regarding the validity or scope of any intellectual
property or other rights that might be claimed to pertain to the implementation
or use of the technology described in this document or the extent to which any
license under such rights might or might not be available; neither does it
represent that it has made any effort to identify any such rights.  Copies of
claims of rights made available for publication and any assurances of licenses
to be made available, or the result of an attempt made to obtain a general
license or permission for the use of such proprietary rights by implementers or
users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights,
patents or patent applications, or other proprietary rights, which may cover
technology that may be required to practice this recommendation.  Please
address the information to the OGF Executive Director.

This document and the information contained herein is provided on an "As Is"
basis and the OGF disclaims all warranties, express or implied, including but
```

```
not limited to any warranty that the use of the information herein will not
infringe any rights or any implied warranties of merchantability or fitness
for a particular purpose.

Copyright (C) Open Grid Forum (2009-2012). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and
derivative works that comment on or otherwise explain it or assist in its
implementation may be prepared, copied, published and distributed, in whole or
in part, without restriction of any kind, provided that the above copyright
notice and this paragraph are included on all such copies and derivative works.
However, this document itself may not be modified in any way, such as by removing
the copyright notice or references to the OGF or other organizations, except as
needed for the purpose of developing Grid Recommendations in which case the
procedures for copyrights defined in the OGF Document process must be followed,
or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by
the OGF or its successors or assignees.

Open Grid Forum NSI Document Distribution Service Protocol v1.0.

Description: This is the NSI Document Distribution Protocol types schema for
the reference web services implementation of the OGF NSI Document Distribution
Service v1.0.  The Document Distribution Service provides the primary mechanism
for information discovery within the Network Service Framework suite of protocols.
Comments and questions can be directed to the mailing list group
mailing list (nsi-wg@ogf.org).
-->
<xsd:schema targetNamespace="http://schemas.ogf.org/nsi/2014/02/discovery/types"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.ogf.org/nsi/2014/02/discovery/types"
    version="1.0">

    <xsd:annotation>
        <xsd:appinfo>ogf_nsi_discovery_protocol_v1_0.xsd 2014-02-20</xsd:appinfo>
        <xsd:documentation xml:lang="en">
            This is an XML schema document describing the OGF NSI Document
            Distribution Service Protocol v1.0.
        </xsd:documentation>
    </xsd:annotation>

    <!-- Collection for root resource definition. -->
    <xsd:element name="collection" type="tns:CollectionType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                This root resource contains a collection of zero or more
                subscriptions and documents held within the NSA.

                HTTP operations: GET
                URI: /

                HTTP Parameters:
                  Accept - Identifies the content type encoding requested for
                  the returned results. Must be a content type supported by the
                  protocol.

                  If-Modified-Since - Return only entries discovered or
                    modified since this time.

                Query Parameters: None

                Returns (code, element):
                    200       collection
                        Return collection element containing all subscription
                        and document resources matching the query.  If no
                        subscriptions or documents match the query, then an empty
                        documents collection is returned.
```

```
                304       None
                    Successful operation where there were no changes to any
                    subscription or document resource given the If-Modified-Since
                    criteria.  Returns no message body.

                400       error
                    Returned if a client specifies an invalid request.  An
                    error element will be included populated with appropriate
                    error information.

                500       error
                    Returned if an internal server error occurred during the
                    processing of this request. An error element will be
                    included populated with appropriate error information.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:complexType name="CollectionType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                Type definition for a collection of discoverable resources.
                This type contains a list of subscriptions and docuemnts
                matching the query parameters.  Extensibility is added to
                allow inclusion of resources from other namespaces as needed.

                Elements:

                subscriptions - A list of subscription resources within the
                system.

                documents - A list of document resources stored within the
                document space of this provider.

                local - A list of document resources published by the local
                provider.

                other - Provides a flexible mechanism allowing additional elements
                to be provided from other namespaces without needing to update
                this schema definition.

                Attributes:

                other - Provides a flexible mechanism allowing additional attributes
                to be provided from other namespaces without needing to update
                this schema definition.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element ref="tns:subscriptions" minOccurs="0" />
            <xsd:element ref="tns:documents" minOccurs="0" />
            <xsd:element ref="tns:local" minOccurs="0" />
            <xsd:any namespace="##other" processContents="lax" minOccurs="0"
                maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:complexType>

    <!-- A list of subscriptions. -->
    <xsd:element name="subscriptions" type="tns:SubscriptionListType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                The subscriptions resource contains a collection of zero or
                more subscriptions held within the provider NSA.

                HTTP operations: GET
                URI: /subscriptions

                HTTP Parameters:
```

```
                Accept - Identifies the content type encoding requested for
                the returned results. Must be a content type supported by the
                protocol.

                If-Modified-Since - Constrains the GET request to return only
                those subscriptions that have been created or updated since the
                time specified in this parameter.

              Query Parameters:
                requesterId - Return all subscription resources containing the
                specified requesterId.

              Returns (code, element):

                200  subscriptions
                      Return all subscription resources matching the query in a
                      subscriptions element.  If no subscriptions match the query,
                      then an empty subscriptions element is returned.

                304   None
                      Successful operation where there were no changes to any
                      subscription resources matching the query filter given the
                      If-Modified-Since criteria. Returns no message body.

                400  error
                      Returned if a DDS requester specifies an invalid request. An error
                      element will be included populated with appropriate error
                      information.

                500  error
                      Returned if an internal server error occurred during the
                      processing of this request. An error element will be included
                      populated with appropriate error information.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:complexType name="SubscriptionListType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                Type definition for a list of subscription resources.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element ref="tns:subscription" minOccurs="0" maxOccurs="unbounded" />
            <xsd:any namespace="##other" processContents="lax" minOccurs="0"
                maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:complexType>

    <!-- A signle subscription resource definition. -->
    <xsd:element name="subscription" type="tns:SubscriptionType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                The subscription resource contains a single subscription from
                the provider NSA.

                HTTP operations: GET
                URI: /subscriptions/{id}
                        {id} is the unique subscription identifier.

                HTTP Parameters:
                Accept - Identifies the content type encoding requested for
                the returned results. Must be a content type supported by the
                protocol.

                If-Modified-Since - Constrains the GET request to return only
                the subscription if it has been updated since the time specified
```

```
                in this parameter.

                Query Parameters: None

                Returns (code, element):

                200   subscription
                    Successful operation returns the subscription identified by
                    id in a subscription element.  The Last-Modified header
                    parameter will contain the time this subscription resource
                    was last modified.

                304 None
                    Successful operation where there were no changes to the
                    subscription resource identified by id given the
                    If-Modified-Since criteria. Returns no message body.

                400   error
                    Returned if a DDS requester specifies an invalid request. An error
                    element will be included populated with appropriate error
                    information.

                404 error
                    Returned if the requested subscription was not found.  An
                    error element will be included populated with appropriate
                    error information.

                500   error
                    Returned if an internal server error occurred during the
                    processing of this request. An error element will be included
                    populated with appropriate error information.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:complexType name="SubscriptionType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                This type models the subscription resource.

                Elements:

                requesterId   - The identifier of the DDS requester that created
                the subscription.  An NSA must use its unique NSA identifier for
                requesterId.

                callback - The HTTP endpoint on the DDS requester that will receive
                the notifications delivered for this subscription.

                filter - The filter criteria to apply to document events to determine
                if a notification should be sent to the DDS requester.

                other - Provides a flexible mechanism allowing additional elements
                to be provided from other namespaces without needing to update
                this schema definition.

                Attributes:

                id - The provider assigned subscription identifier.

                href - The direct URI reference to the resource.

                version - The version of the subscription.  Indicates the last
                time the subscription was modified.

                other - Provides a flexible mechanism allowing additional attributes
                to be provided from other namespaces without needing to update
                this schema definition.
            </xsd:documentation>
```

```
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="requesterId" type="xsd:string" />
            <xsd:element name="callback"    type="xsd:anyURI" />
            <xsd:element name="filter"      type="tns:FilterType" minOccurs="0" />
            <xsd:any namespace="##other" processContents="lax" minOccurs="0"
                maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="id"      use="required" type="xsd:string" />
        <xsd:attribute name="href"    use="required" type="xsd:anyURI" />
        <xsd:attribute name="version" use="required" type="xsd:dateTime" />
        <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:complexType>

    <xsd:element name="subscriptionRequest" type="tns:SubscriptionRequestType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                The subscriptionRequest is a collection of parameters from the
                subscription resource that is used to create a new subscription
                resource or update an existing subscription resource.

                Once a subscription has been successfully created or updated on
                the provider the server will immediately send notifications for
                all documents matching the filter criteria independent of the
                event filter.

                HTTP operations: POST (create), PUT (update)
                URI: /subscriptions

                HTTP Parameters:
                Content-Type - Identifies the content type encoding of the POST
                body contents.  Must be a content type supported by the protocol.

                Accept - Identifies the content type encoding requested for
                the returned results. Must be a content type supported by the
                protocol.

                If-Modified-Since - Constrains the GET request to return only
                the subscription if it has been updated since the time specified
                in this parameter.

                Query Parameters: N/A

                Returns (code, element):

                201   subscription
                    Returns a copy of the new subscription resource created as
                    the result of a successful operation.  The HTTP Location
                    header field will contain the URI of the new subscription
                    resource.

                400   error
                    Returned if a DDS requester specifies an invalid request. An error
                    element will be included populated with appropriate error
                    information.

                403   error
                    The server understood the request, but is refusing to fulfill
                    it. Authorization will not help and the request SHOULD NOT be
                    repeated.  An error element will be included populated with
                    appropriate error information.

                500   error
                    Returned if an internal server error occurred during the
                    processing of this request. An error element will be included
                    populated with appropriate error information.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>
```

```
<xsd:complexType name="SubscriptionRequestType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            This type models a subset of parameters from the subscription
            resource used during creation and updates.

            Elements:

            requesterId   - The identifier the DDS requester would like to
            use for unique identification.  An NSA must use its unique NSA
            identifier for requesterId.

            callback - The HTTP endpoint on the DDS requester that will receive
            the notifications delivered for this subscription.

            filter - The filter criteria to apply to document events to determine
            if a notification should be sent to the DDS requester.

            other - Provides a flexible mechanism allowing additional elements
            to be provided from other namespaces without needing to update
            this schema definition.

            Attributes:

            other - Provides a flexible mechanism allowing additional attributes
            to be provided from other namespaces without needing to update
            this schema definition.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="requesterId" type="xsd:string" />
        <xsd:element name="callback"    type="xsd:anyURI" />
        <xsd:element name="filter"      type="tns:FilterType" minOccurs="0" />
        <xsd:any namespace="##other" processContents="lax" minOccurs="0"
            maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:complexType name="FilterType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            This type is the base notification filter for subscriptions.
            The include element specifies the document event match criteria
            to include, while the exclude element specifies those to
            specifically exclude.  The include will be evaluated first, then
            the exclude will be applied.

            Elements:

            include - Include notifications matching these criteria.

            exclude - Exclude the notifications matching these criteria.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="include" type="tns:FilterCriteriaType" minOccurs="0"
            maxOccurs="unbounded" />
        <xsd:element name="exclude" type="tns:FilterCriteriaType" minOccurs="0"
            maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="FilterCriteriaType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            This type models the criteria that can be included in the
            notfication filter for subscriptions.
```

```
            Elements:

            event - The type of document event that will generate a
            notification.  Currently only three events are supported (All,
            New, Updated).  At least one of event criteria must be
            supplied.  The default event criteria is All.

            or - Any document matching any of the supplied nsa, document
            type, or document id values.

            and - Any document matching all of the supplied nsa, document
            type, or document id values (logical AND).
        </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="event" type="tns:DocumentEventType" default="All"
            minOccurs="1" maxOccurs="3" />
        <xsd:element name="or"    type="tns:FilterOrType" minOccurs="0"
            maxOccurs="unbounded" />
        <xsd:element name="and"   type="tns:FilterAndType" minOccurs="0"
            maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="DocumentEventType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            This is a simple string type enumerating the types of document
            events that can be included in a filter.

            All - Matches all document events.

            New - Matches new documents that are discovered in the space.

            Updated - Matches existing documents in the space that are updated.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="All"/>
        <xsd:enumeration value="New"/>
        <xsd:enumeration value="Updated"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="FilterAndType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            This filter criteria type lists elements that can be matched in a
            document as part of the decision to generate or not generate a
            notification.  The supplied nsa, document type, and document id
            values are evaluted as a logical AND so that all included values
            must match.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="nsa"  type="xsd:anyURI" minOccurs="0" />
        <xsd:element name="type" type="xsd:string" minOccurs="0" />
        <xsd:element name="id"   type="xsd:string" minOccurs="0" />
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="FilterOrType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            This filter criteria type lists elements that can be matched in a
            document as part of the decision to generate or not generate a
            notification.  The supplied nsa, document type, and document id
            values are evaluted as a logical OR so that any included values
```

```
            that match result in a criteria match.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:choice maxOccurs="unbounded">
            <xsd:element name="nsa"  type="xsd:anyURI" />
            <xsd:element name="type" type="xsd:string" />
            <xsd:element name="id"   type="xsd:string" />
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>

<!-- A list of notifications. -->
<xsd:element name="notifications" type="tns:NotificationListType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            When a document event occurs matching a registered subscription
            the provider must issue a notification to the requester endpoint
            identified in the subscription resource.  This element is sent
            in the body of a POST request to the requester endpoint.

            Multiple events can be grouped and delivered together in a single
            notification if these events occur within a reasonable period of
            time of each other. Notification delivery should not be delayed.

            Notifications are also sent when a subscription is first created,
            and after a subscription is modified.  This notification will
            include any documents matching the filter criteria.

            HTTP operations: POST
            URI: /requester-supplied-endpoint

            HTTP Parameters:

            Content-Type - Identifies the content type encoding of the POST
            body contents.  Must be identical to the value as used by the
            DDS requester on subscription.

            Query Parameters: N/A

            Returns (code, element):

            202    None
                Indicates the subscribed DDS requester has accepted the notification
                for processing.  The DDS requester receiving the notification must
                return an HTTP 202 status code in response to the POST.
                Any other status code will result in a deletion of the
                subscription.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:complexType name="NotificationListType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            Type definition for a list of notifications.

            Elements:

            notification - A list of zero or more notifications matching the
            subscription filter criteria.

            Attributes:

            providerId - The identifier of the provider generating the
            notification.  This is the provider on which the subscription
            was created.

            id - The identifier of the subscription that generated the
```

```
            notifications.

            href - The URI reference for subscription that generated the
            notification.  This can be used to directly access the
            subscription.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element ref="tns:notification" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="providerId" use="required" type="xsd:anyURI" />
    <xsd:attribute name="id"         use="required" type="xsd:string" />
    <xsd:attribute name="href"       use="required" type="xsd:anyURI" />
</xsd:complexType>

<!-- A single notfication. -->
<xsd:element name="notification" type="tns:NotificationType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            This element models a single document notification and is
            included in the notifications element.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:complexType name="NotificationType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            This type models a single document notification event.

            Elements:

            discovered - The time this document event was detected on the
            provider.  It is not the time the notification was generated.
            It also should be noted that this time could be a considerable
            period in the past if the notification was sent as the result
            of a subscription creation or edit.

            event - The type of document event this notification represents.

            document - The document metadata entry associated with the
            notification.

            other - Provides a flexible mechanism allowing additional element
            to be provided from other namespaces without needing to update
            this schema definition.

            Attributes:

            other - Provides a flexible mechanism allowing additional attributes
            to be provided from other namespaces without needing to update
            this schema definition.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="discovered" type="xsd:dateTime" />
        <xsd:element name="event"      type="tns:DocumentEventType" />
        <xsd:element name="document"   type="tns:DocumentType" />
        <xsd:any namespace="##other" processContents="lax" minOccurs="0"
            maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<!-- A list of documents. -->
<xsd:element name="documents" type="tns:DocumentListType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            The documents element models a list of documents from the
```

```
document space.

HTTP operations: GET
URI: /documents/{nsa}/{type}

The documents element contains document resources discovered
within the document space, or a subset of documents based on
supplied query parameters.  Zero or more document instances will
be returned in a documents element.

The URI template "/documents/{nsa}/{type}" can be used as an
alternative to, or in conjunction with, the use of query
parameters.  Performing a GET on "/documents/{nsa}/" will
return all documents associated with the specified NSA.
Performing a GET on "/documents/{nsa}/{type}" will return
all documents of {type} from the specified NSA.

HTTP Parameters:

Accept - Identifies the content type encoding requested for
the returned results. Must be a content type supported by the
protocol.

If-Modified-Since - Constrains the GET request to return only
those documents that have been created or updated since the
time specified in this parameter.

Query Parameters:

id (string) - Return all document resources containing the specified Id.

nsa (string) - Return all document resources containing the
specified nsa identifier.  Cannot be used if the {nsa} URI
component is provided.

type (string) - Return all document resources containing the
specified type. Cannot be used if the {type} URI component is
provided.

summary       (none) - Will return summary results of any documents
matching the query criteria.  Summary results includes all
document meta-data but not the signature or document content.

Returns (code, element):

200   documents
Return all document resources matching the query in a
documents element.  If no documents match the query,
then an empty documents element is returned.

304   None
Successful operation where there were no changes to any
subscription resources matching the query filter given the
If-Modified-Since criteria. Returns no message body.

400   error
Returned if a DDS requester specifies an invalid request. An error
element will be included populated with appropriate error
information.

500   error
Returned if an internal server error occurred during the
processing of this request. An error element will be included
populated with appropriate error information.

HTTP operations: POST
URI: /documents

The POST operation on the "/documents" resource will create a
```

```
                new document using the information supplied in the document
                element contained in the POST body.  A successful operation
                will return the new document resource.  This operation has
                restricted access for DDS requesters and is made available by the
                server based on access control permissions.

                Once a document has been successfully created on the server,
                the server will immediately send notifications to all
                subscriptions with filter criteria matching the document.

                HTTP Parameters:

                Content-Type - Identifies the content type encoding of the POST
                body contents.  Must be a content type supported by the protocol.

                Accept - Identifies the content type encoding requested for
                the returned results. Must be a content type supported by the
                protocol.

                If-Modified-Since - Constrains the GET request to return only
                those documents that have been created or updated since the
                time specified in this parameter.

                Body Parameters:

                document - The document to add to the document space of the
                local provider.

                Returns (code, element):

                201   document
                    Returns a copy of the new document resource created as the
                    result of a successful operation.  The HTTP Location header
                    field will contain the direct URI reference of the new
                    document resource.  It will be structured using the URI
                    template $root/documents/{nsa}/{type}/{id}.

                400   error
                    Returned if a DDS requester specifies an invalid request.  An error
                    element will be included populated with appropriate error
                    information.

                403   error
                    The server understood the request, but is refusing to fulfill
                    it. Authorization will not help and the request SHOULD NOT
                    be repeated.  An error element will be included populated
                    with appropriate error information.

                409   error
                    A document already exists with the same name (nsa/type/id).
                    An update of an existing document should use the PUT
                    operation.

                500   error
                    Returned if an internal server error occurred during the
                    processing of this request. An error element will be
                    included populated with appropriate error information.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:element name="local" type="tns:DocumentListType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                The local element models a list of documents from the document
                space published by the local provider NSA.

                HTTP operations: GET
                URI: /local/{type}
```

```
                The local element contains document resources published by the
                local provider, or a subset of documents based on supplied query
                parameters.  Zero or more document instances will be returned in
                a local element.

                A DDS requester can perform a GET operation on the special "/local" URI
                when it would like to discover all documents associated with the
                local provider NSA.  This operation is equivalent to performing a
                GET operation on the URI "/documents/{nsa}", however, for "/local"
                the DDS requester is not required to have previous knowledge of the
                provider NSA identifier.

                The URI template "/local/{type}" can be used as an alternative to,
                or in conjunction with, the use of query parameters.  Performing
                a GET on "/local/{type}/" will return all documents of {type}
                associated with the local NSA.

                HTTP Parameters:

                Accept - Identifies the content type encoding requested for
                the returned results. Must be a content type supported by the
                protocol.

                If-Modified-Since - Constrains the GET request to return only
                those documents that have been created or updated since the
                time specified in this parameter.

                Query Parameters:

                id (string) - Return all document resources containing the
                specified Id.

                type (string) - Return all document resources containing the
                specified type. Cannot be used if the {type} URI component is
                provided.

                summary       (none) - Will return summary results of any documents
                matching the query criteria.  Summary results includes all
                document meta-data but not the signature or document content.

                Returns (code, element):

                200   local
                    Return all document resources matching the query in a
                    documents element.  If no documents match the query,
                    then an empty documents element is returned.

                304   None
                    Successful operation where there were no changes to any
                    document resources matching the query filter given the
                    If-Modified-Since criteria. Returns no message body.

                400   error
                    Returned if a DDS requester specifies an invalid request. An error
                    element will be included populated with appropriate error
                    information.

                500   error
                    Returned if an internal server error occurred during the
                    processing of this request. An error element will be included
                    populated with appropriate error information.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:complexType name="DocumentListType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
```

```
            This type provides a list of zero or more documents.

            Elements:

            document - The document meta-data entry within the document space.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element ref="tns:document" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:complexType>

<!-- A single document. -->
<xsd:element name="document" type="tns:DocumentType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            The document element models the metadata for a single document
            from the document space.

            HTTP operations: GET
            URI: /documents/{nsa}/{type}/{id}

            This operation will return a specific document instance
            discovered within the document space based on the URI template
            "/documents/{nsa}/{type}/{id}", where {nsa} is the NSA sourcing
            the document, {type} is the type of document, and {id} is the
            identifier of the specific document.  The matching document is
            returned in a single document element.

            HTTP Parameters:

            Accept - Identifies the content type encoding requested for
            the returned results. Must be a content type supported by the
            protocol.

            If-Modified-Since - Constrains the GET request to return only
            those documents that have been created or updated since the
            time specified in this parameter.

            Query Parameters: None.

            Returns (code, element):

            200   local
                Successful operation returns the document identified by
                {nsa}/{type}/{id} in a document element.  The Last-Modified
                header parameter will contain the time this document resource
                was last discovered.

            304   None
                Successful operation returns the document identified by
                {nsa}/{type}/{id} in a document element.  The Last-Modified
                header parameter will contain the time this document resource
                was last discovered.

            400   error
                Returned if a DDS requester specifies an invalid request. An error
                element will be included populated with appropriate error
                information.

            404   error
                Returned if the requested document was not found.  An error
                element will be included populated with appropriate error
                information.

            500   error
                Returned if an internal server error occurred during the
                processing of this request. An error element will be included
                populated with appropriate error information.
```

```
                HTTP operations: PUT
                URI: /documents/{nsa}/{type}/{id}

                The PUT operation on the "/documents/{nsa}/{type}/{id}" resource
                will allow a DDS requester to edit the document corresponding to the
                identifier {id}, using the information supplied in the document
                element contained in the PUT body.  A successful operation will
                return the modified document and trigger any associated
                notifications within the NSA.

                A document is deleted from the document space by updating it's
                expire date to a reasonably short period in the future.  This
                updated document will get propagated throughout the document
                space and then expire, removing it from the space.

                 HTTP Parameters:

                 Content-Type - Identifies the content type encoding of the PUT
                 body contents.  Must be a content type supported by the
                 protocol.

                 Accept - Identifies the content type encoding requested for
                 the returned results. Must be a content type supported by the
                 protocol.

                 Body Parameters:

                 document - The document to update in the document space of the
                 local provider. The PUT request must contain the document
                 element containing the existing parameters of the document
                 resource if they were not modified, as well as any new/edited
                 values.

                 Returns (code, element):

                 200 document
                    Returns a copy of the modified document resource as the
                    result of a successful operation.

                 400 error
                    Returned if a DDS requester specifies an invalid request.  An
                    error element will be included populated with appropriate
                    error information.

                 403 error
                    The server understood the request, but is refusing to fulfill
                    it. Authorization will not help and the request SHOULD NOT be
                    repeated.  An error element will be included populated with
                    appropriate error information.

                 404 error
                    Returned if the requested document was not found.  An error
                    element will be included populated with appropriate error
                    information.

                 500 error
                    Returned if an internal server error occurred during the
                    processing of this request. An error element will be included
                    populated with appropriate error information.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:complexType name="DocumentType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                The DocumentType type definition models all data relating to
                a single document exchanged within the network.  Meta-data
```

```
associated with the document, document signature, and the
document itself is encapsulated in this type.  The type
itself is structured such that it does not need to be
manipulated between receiving and propagating to a peer.

A document is uniquely named within the network by the tuple
of nsa, type, and id.  The identifier (id) element itself does
not need to be unique within the network; it must just be unique
within the context of the nsa and type elements.  These rules
allow the reuse of the same id value for a document of different
types under the same source NSA.  This is important for both
searching, and for associating the same naming attribute to
related documents.

An NSA must not modify the content of a DocumentType before
propagating on to a peer unless that NSA is the owner of the
document.

Elements:

nsa    - The source NSA associated with the generation and management
of the document within the network. This is assumed to be the NSA
to which the document relates, however, there may be situations
such as proxy publishing where this assumption is not true.

For example, if the document being generated is the NSA Description
Document for NSA “urn:ogf:network:example.com:2013:nsa:vixen”, then
the nsa element should contain is the NSA identifier
“urn:ogf:network:example.com:2013:nsa:vixen”.

type - The unique string identifying the type of this document.
A document type is defined by the type and release of a data
document.  For example, NSI Topology version 1.0 and a NSI
Topology version 2.0 would be considered two different document
types:
    - vnd.ogf.nsi.topology.v1+xml
    - vnd.ogf.nsi.topology.v2+xml

The NSA Description Document 1.0 is defined as the type:
    - vnd.ogf.nsi.nsa.v1+xml

signature - The OPTIONAL digital signature of the document
content.

content - The content of the document modeled by this document
resource.  The document containted in this element must be
encoded as a MIMW string following the content transfer encoding
rules as defined in RFC1341.

other - Provides a flexible mechanism allowing additional elements
to be provided from other namespaces without needing to update
this schema definition.

Attributes:

id - The identifier of the document.  This value must be unique
in the context of the nsa and type element values within the
global document space.

version       - The version of the document, or more specifically, the
date this version of the document was created.  Any updates to the
document must be tagged with a new version.

expires       - The date this version of the document expires and
should be deleted from the Global Document Space by an NSA and
any DDS requesters caching the document.

other - Provides a flexible mechanism allowing additional attributes
to be provided from other namespaces without needing to update
```

```
            this schema definition.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="nsa"       type="xsd:anyURI" />
        <xsd:element name="type"      type="xsd:string" />
        <xsd:element name="signature" type="tns:ContentType" minOccurs="0" />
        <xsd:element name="content"   type="tns:ContentType" minOccurs="0" />
        <xsd:any namespace="##other"  processContents="lax" minOccurs="0"
            maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="id"      use="required" type="xsd:string" />
    <xsd:attribute name="href"    use="optional" type="xsd:anyURI" />
    <xsd:attribute name="version" use="required" type="xsd:dateTime" />
    <xsd:attribute name="expires" use="required" type="xsd:dateTime" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:complexType name="ContentType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            This simple string type is used to hold a document contents or
            digital signature within the document metadata.  Elements of
            is type use the contentTransferEncoding and contentType
            attributes to describe the encoding of the document within
            this string value.  The document meta-data "type" element
            identifies the document type itself.

            When encoding a document to be contained in this element, the
            contentType attribute is applied first using rules defined in
            RFC1341 (section 4), followed by the contentTransferEncoding
            attribute using rules defined in RFC1341 (section 5).  As an
            example, an NSI topology document version 2 is encoded as
            follows:

            type="vnd.ogf.nsi.topology.v2+xml"
            contentType="application/x-gzip"
            contentTransferEncoding="base64"

            In this case the "vnd.ogf.nsi.topology.v2+xml" document type
            (XML) is compressed using gzip into a binary encoding, then
            base64 encoded before being stored in the content element for
            addition to the DDS.

            When decoding the contents contained in this element, the
            contentTransferEncoding attribute is applied first using rules
            defined in RFC1341 (section 5), followed by the contentType
            attribute using rules defined in RFC1341 (section 4).  As an
            example, an NSI Description Document version 1 is encoded as
            follows:

            type="vnd.ogf.nsi.nsa.v1+xml"
            contentType="application/x-gzip"
            contentTransferEncoding="base64"

            In this case the "vnd.ogf.nsi.nsa.v1+xml" document type
            (XML) will need to decoded from base64 as indicated by the
            contentTransferEncoding attribute, then decompressed using
            gzip from the binary encoding into the resulting XML as
            specified by the type.

            Attributes:

            contentType - This attribute is used to specify the nature
            of the data in the body of the content element, by giving type
            and subtype identifiers, and by providing auxiliary information
            that may be required for certain document types.  RFC1341
            (section 4) describes this in more detail.
```

```
                contentTransferEncoding - This attribute is used to indicate
                the type of transformation that has been used in order
                to represent the body in an acceptable manner for transport in
                the string content element of the document meta-data.  The
                supported values of this attribute are defined in RFC1341
                (section 5).
            </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="contentType" use="optional" type="xsd:string" />
                <xsd:attribute name="contentTransferEncoding" use="optional"
                    type="xsd:string" />
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:element name="error" type="tns:ErrorType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                The error element is returned in an HTTP response when an error
                has occured servicing the request on the provider.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:complexType name="ErrorType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                This type models errors returned from Document Distribution
                Service operations.

                Elements:

                code - The integer error code for the specific error.

                label - A character string label for the error.

                description - A detailed description of error.

                resource - The resource that caused the error.

                Attributes:

                id - The unique identifier of the error for correlation with logs.

                date - The date and time the error occured.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="code"        type="xsd:int" />
            <xsd:element name="label"       type="xsd:string" />
            <xsd:element name="description" type="xsd:string" />
            <xsd:element name="resource"    type="xsd:anyURI" />
        </xsd:sequence>
        <xsd:attribute name="id"   use="required" type="xsd:string" />
        <xsd:attribute name="date" use="required" type="xsd:dateTime" />
    </xsd:complexType>
</xsd:schema>
```