Open Archives Initiative
ResourceSync Framework Specification

ResourceSync Framework Specification - Change Notification

20 July 2017

This version:
http://www.openarchives.org/rs/notification/1.0.1/notification
Latest version:
http://www.openarchives.org/rs/notification
Previous version:
http://www.openarchives.org/rs/notification/1.0/notification

Editors:
Martin Klein, Herbert Van de Sompel - Los Alamos National Laboratory
Simeon Warner - Cornell University
Graham Klyne - University of Oxford
Bernhard Haslhofer - University of Vienna
Michael Nelson - Old Dominion University
Carl Lagoze - University of Michigan
Robert Sanderson - The Getty

Abstract

The ResourceSync core specification introduces a pull-based synchronization framework for the web that consists of various capabilities that a Source can implement to allow Destinations to remain synchronized with its evolving resources. This ResourceSync Change Notification specification describes an additional, push-based, capability that a Source can support. It is aimed at reducing synchronization latency and entails a Source sending notifications to subscribing Destinations.

Status of this Document

This specification is one of several documents comprising the ResourceSync Framework Specifications. Feedback is most welcome on the ResourceSync Google Group.

Table of Contents

1. Introduction

This specification describes a Change Notification capability defined for the ResourceSync framework. The push-based notification capability is aimed at decreasing the synchronization latency between a Source and a Destination that is inherent in the pull-based capabilities defined in the ResourceSync core specification. The Change Notification capability consists of a Source sending notifications about changes to its resources, for example the creation or deletion of a resource. Another specification describes a Framework Notification capability that consists of a Source sending out notifications about changes to its implementation of the ResourceSync framework, for example the publication of a new Resource List or the updating of a Change List.

1.1. Motivating Example

Applications based on Linked Data integrate resources from various datasets, with resources likely changing at a different pace. The BBC Linked Data applications that integrate data from, among others, Last.FM, DBpedia, MusicBrainz, and GeoNames serve as examples. The accuracy of services based on such an integrated resource collection depends on the contributing resources being up-to-date. The update frequency of LiveDBPedia resources, for example, has been observed to average around two changes per second. This provides a significant synchronization challenge that the Change Notification capability aims to address.

1.2. Definitions and Namespace Prefix Bindings

This specification uses the terms "resource", "representation", "request", "response", "content negotiation", "client", and "server" as described in Architecture of the World Wide Web.

Throughout this document, the following namespace prefix bindings are used:

Prefix	Namespace URI	Description
none	`http://www.sitemaps.org/schemas/sitemap/0.9`	Sitemap XML elements defined in the Sitemap protocol
`rs`	`http://www.openarchives.org/rs/terms/`	Namespace for elements and attributes introduced in this specification

Table 1: Namespace prefix bindings used in this document

2. Change Notification Channels

Change Notifications are sent to inform Destinations about resource change events, specifically, when a Source's resource that is subject to synchronization was created, updated, or deleted. The payload for these notifications is described in Section 3. Notifications are sent from Source to Destination on one or more channels provided by a push technology discussed in Section 4.

Figure 1 displays the structure of the ResourceSync framework for a Source that has a single set of resources, showing the Source Description and the Capability List at the top. The Capability List advertises four distinct capabilities: a Resource List, a Change List, a Resource Dump, and a Change Dump. The figure also shows a Change Notification channel (yellow hexagon) and indicates it is used to send information about resource changes for a specific set of resources. Changes to these resources are communicated as change notifications via the Change Notification channel.

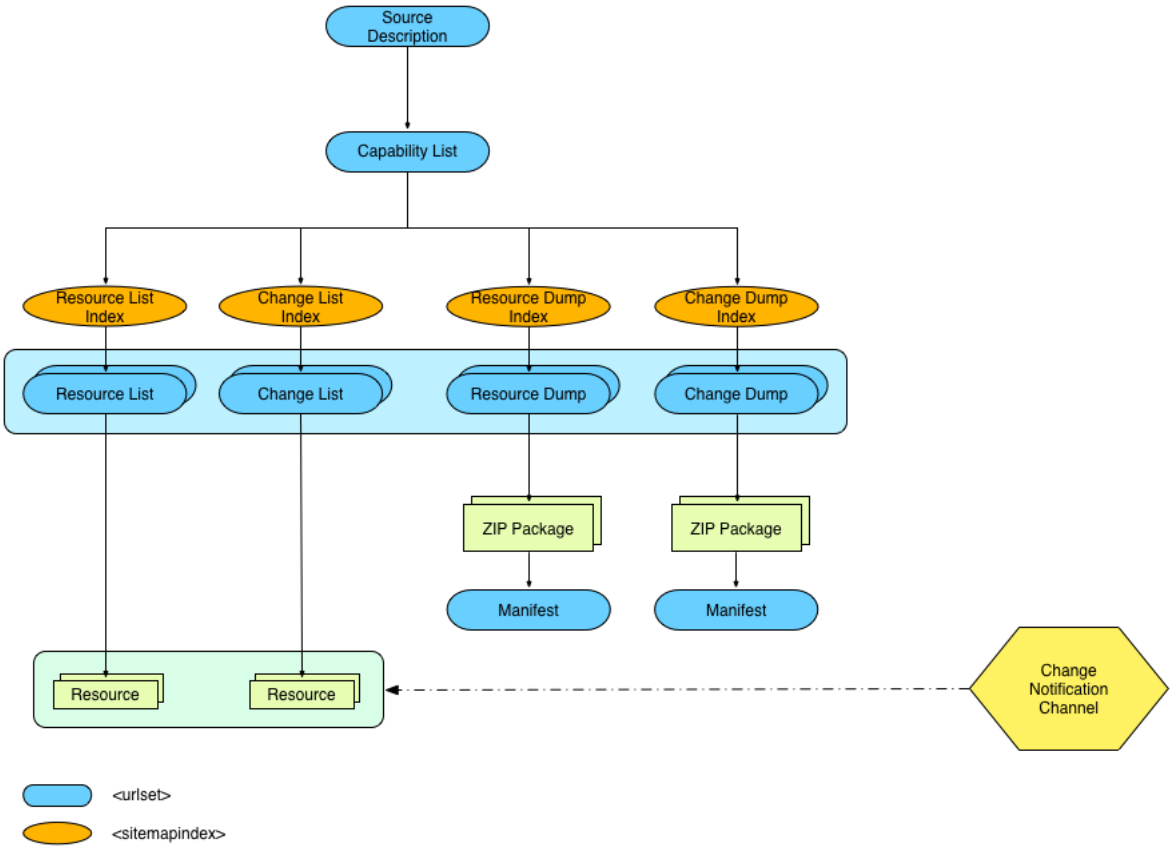

Figure 1: A Change Notification channel in the ResourceSync framework structure

The ResourceSync framework allows a Source to offer multiple sets of resources in which case the Source Description points to multiple Capability Lists, one for each set of resources. In this case, a dedicated Change Notification channel must be provided for each distinct set of resources for which Change Notification is supported. A notification about a change to a resource is sent via the Change Notification channel that is associated with the set of resources under which the resource resides. If a resource resides under multiple sets of resources, a notification is sent on each of the Change Notification channels associated with those sets of resources. Change Notifications must be sent on different channels.

Figure 2 depicts a scenario where a Source offers multiple sets of resources and its Source Description therefore points to multiple Capability Lists, one for each set of resources, in this case `Capability List 1` and `Capability List 2`. Figure 2 shows that each set of resources has a designated Change Notification channel. `Change Notification Channel 1`, for example, is used to send change notifications about changes to resources that are part of the set of resources covered by `Capability List 1`.

Note that the creation and deletion of Change Notification channels is reflected in updated Capability Lists (see Section 5). This specification does not define a separate notification about notification channels.

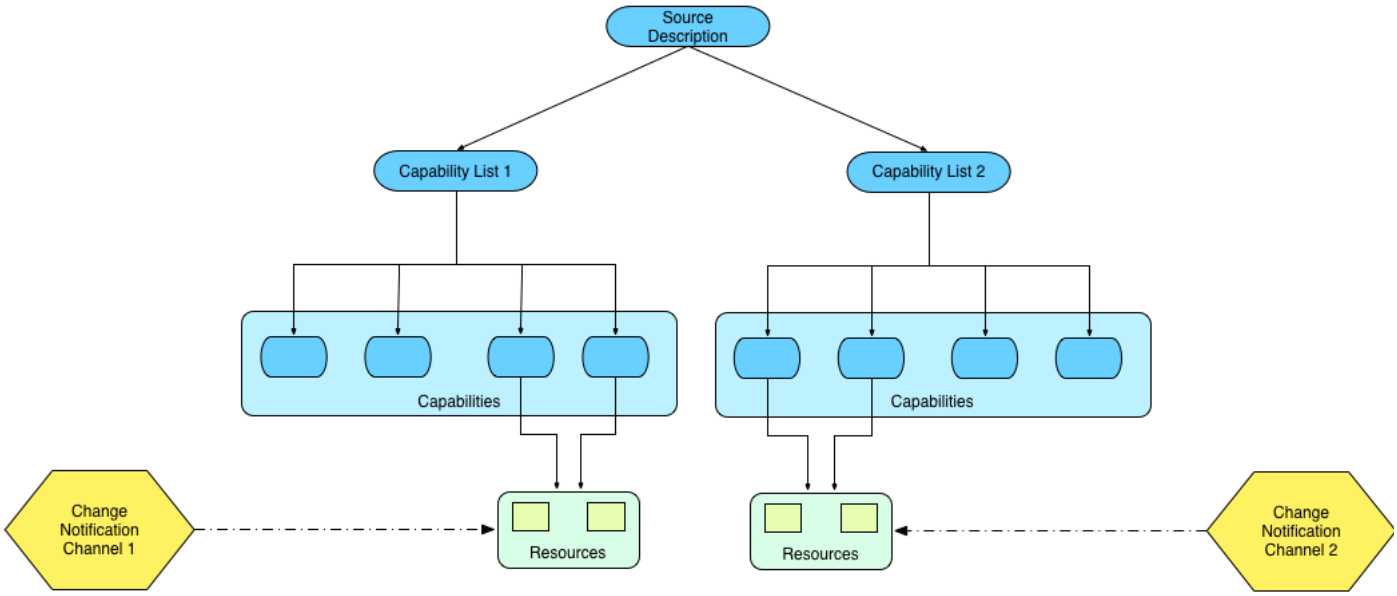

Figure 2: Change Notification channels for multiple sets of resources

3. Change Notification

A change notification is sent on the appropriate Change Notification channel, as described in Section 2, if a Source wishes to notify a Destination that one or more of its resources subject to synchronization have changed. By subscribing to a Change Notification channel, a Destination can reduce synchronization latency and avoid periodically polling the Source's Change Lists - if they exist - to determine whether resource changes have occurred.

The format of a change notification is very similar to the Change List format introduced in Section 12 of the core specification. All entries in a change notification must be provided in forward chronological order: the resource with the least recent change datetime must be listed at the beginning of the change notification payload, while the resource with the most recent change datetime must be listed at the end. The format is based on the `<urlset>` document format introduced by the Sitemap protocol. It has the `<urlset>` root element and the following structure:

- The mandatory `<rs:md>` child element of `<urlset>` must have a `capability` attribute that has a value of `change-notification`. It also has the mandatory `from` and `until` attributes. The `from` attribute indicates that the change notification includes all changes that occurred to the set of resources at the Source since the datetime expressed as the value of the attribute. The `until` attribute indicates that the change notification includes all changes that occurred to the set of resources at the Source up until the datetime expressed as the value of the attribute. Consecutive change notifications sent by the Source must cover consecutive and non-overlapping temporal `from`/`until` intervals. This allows a Destination to verify whether it received all change notifications and to process them in the appropriate order in case they were not received in the order the Source sent them.
- The optional `<rs:ln>` child element of `<urlset>` points to the Capability List with the relation type `up`.
- One `<url>` child element of `<urlset>` for each resource change. This element does not have attributes, but uses child elements to convey information about the changed resource. The `<url>` element has the following child elements:
  - A mandatory `<loc>` child element provides the URI of the changed resource.
  - An optional `<lastmod>` child element with semantics as described in Section 7 of the core specification.
  - A mandatory `<rs:md>` child element must have the attribute `change` to convey the nature of the change. Its value can be `created`, `updated`, or `deleted`. It has the optional `datetime` attribute to convey the datetime of the change, as described in Section 7 of the core specification. The `<rs:md>` child element can further have attributes `hash`, `length`, and `type`, as described in Section 7 of the core specification.
  - Optional `<rs:ln>` child elements link to related resources as described in Section 7 and Section 14 of the core specification.

Change notifications do not use the `<sitemapindex>` document format introduced by the Sitemap protocol. In the event that there are a very large number of simultaneous changes at a Source, the notifications must be split into a sequence of change notifications using `<urlset>` documents.

Example 1 shows the payload of a change notification containing the description of changes to two resources.

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
        xmlns:rs="http://www.openarchives.org/rs/terms/">
  <rs:ln rel="up"
         href="http://example.com/dataset1/capabilitylist.xml"/>
  <rs:md capability="change-notification"
         from="2013-01-03T00:00:00Z"
         until="2013-01-03T00:10:00Z"/>
  <url>
      <loc>http://example.com/res1</loc>
      <lastmod>2013-01-01T13:03:00Z</lastmod>
      <rs:md change="created"
             datetime="2013-01-03T00:07:22Z"
             hash="md5:1584abdf8ebdc9802ac0c6a7402c03b6"
             length="8876"
             type="application/pdf"/>
   </url>
   <url>
```

```
      <loc>http://example.com/res2</loc>
      <rs:md change="updated"
             datetime="2013-01-03T00:08:52Z"
             hash="md5:1e0d5cb8ef6ba40c99b14c0237be735e
             sha-256:854f61290e2e197a11bc91063afce22e43f8ccc655237050ace766adc68dc784"
             length="14599"
             type="text/html"/>
   </url>
</urlset>
```

Example 1: The payload of a change notification

4. Transport Protocol: WebSub

In order to bootstrap the notification capabilities of the ResourceSync framework, a single transport protocol is chosen: WebSub. WebSub is a simple, HTTP-based publish/subscribe protocol that is expected to perform well for use cases that do not require change notifications to be sent at a very high frequency.

The below description of the use of WebSub for Change Notifications is essentially the same as the description of its use for Framework Notifications. However, in the interest of self-containdeness of the respective specifications, and because Change Notifications and Framework Notifications must be sent on different channels, the description is repeated in both specifications.

Table 2 maps terminology used in ResourceSync and WebSub. In order to implement the publish/subscribe paradigm, WebSub introduces a `hub` that acts as a conduit between Source and Destination. A hub can be operated by the Source itself or by a third party. It is uniquely identified by the `hub URI`. WebSub's `topic` corresponds with the notion of channel used in this specification. A topic is uniquely identified by its `topic URI`. Hence, per set of resources, the Source has a dedicated topic (and hence topic URI) for change notifications.

ResourceSync	WebSub
Source	Publisher
Destination	Subscriber
Channel	Topic
Notification	Notification
	Hub

Table 2: Mapping of terminologies between ResourceSync and WebSub

The remainder of this section describes the use of WebSub in ResourceSync. It only provides the information about the WebSub protocol that is essential to gain an adequate understanding of the overall mechanism. Details about the WebSub protocol are available in the WebSub specification. Figure 3 shows an overview of HTTP interactions between Source, Hub, and Destination. They will be detailed in the remainder of this section.

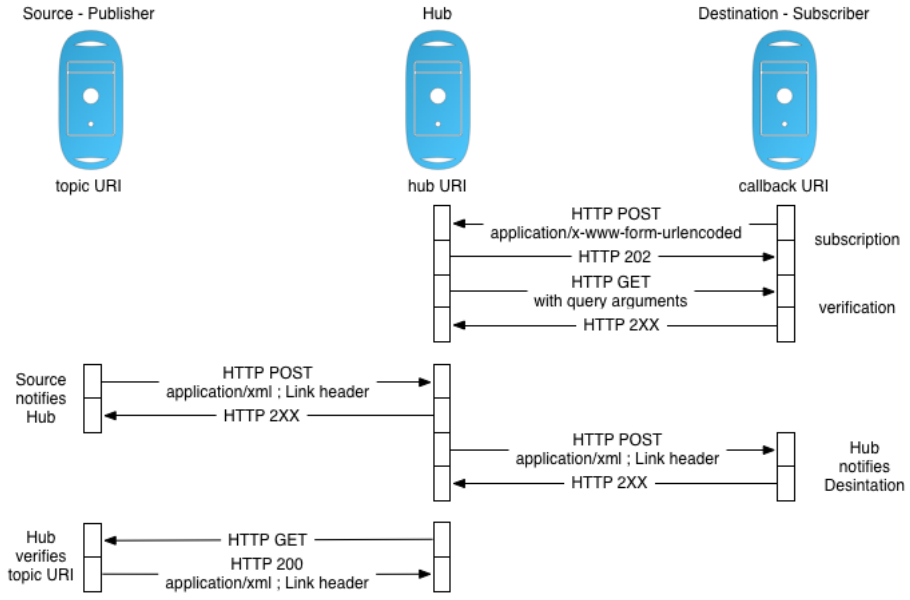

Figure 3: HTTP interactions between Source, Hub, and Destination

4.1. Source Submits Notifications to Hub

The WebSub protocol provides no specific guidelines regarding the way in which a Source should communicate notifications to a hub. The mechanism for ResourceSync change notifications is as follows:

- For each set of resources, the Source has a dedicated topic URI for its change notifications. The response to an HTTP HEAD/GET request issued against a topic URI must have an HTTP "200 OK" response code, an HTTP `Content-Type` header with a value of `application/xml`, and an HTTP Link header with the following links:
  - A mandatory link with the `self` relation type that provides the topic URI for the submitted notification as the value of the `href` attribute.
  - A mandatory link with the `hub` relation type that provides the hub URI as the value of the `href` attribute.

  The resource identified by the topic URI must have the same `Content-Type` (`application/xml`) as the payload of the notification. If the intention is to serve content at the topic URI the payload of the most recent notification may be suitable. If no content is provided at the topic URI the response header `Content-Length` with the value `0` should be returned. This communication is shown as "Hub verifies topic URI" in Figure 3.
- A Source submits the payload for a change notification to a hub. The notification payload is provided as the body of an HTTP POST issued against the hub URI. This HTTP POST must have an HTTP `Content-Type` header with a value of `application/xml` and an HTTP Link header with the following links:
  - A mandatory link with the `self` relation type that provides the topic URI for the submitted notification as the value of the `href` attribute.
  - A mandatory link with the `hub` relation type that provides the hub URI as the value of the `href` attribute.

  This is shown as the right pointing arrow of "Source notifies Hub" in Figure 3.
- The hub, if operated by a third party, may want to verify whether the HTTP POST was submitted by a legitimate Source. The hub can do so by establishing a trust relationship with the Source prior to relaying any of its notifications, for example, through a one time registration mechanism. Once the Hub is relaying notifications, it can still check whether the Source adheres to the WebSub protocol. This is the case if all the following hold:
  - The HTTP POST contains an HTTP Link header with the aformentioned links.
  - The response to an HTTP HEAD/GET issued against the topic URI contains an HTTP Link header with the aformentioned links.
- The hub responds to the submission of the payload for a change notification as follows:
  - The response must have HTTP "200 OK" status code if the submission was successful.
  - The response must have a HTTP 3XX, 4XX or 5XX status code if the submission was not successful.

  This is shown as the left pointing arrow of "Source notifies Hub" in Figure 3.

Example 2 shows the HTTP POST issued by the Source against its hub to submit the change notification payload of Example 1. For brevity, the payload is not shown in its entirety. The third party hub URI is `http://hub.example.org/websub/` and the Source's topic URI (channel) for change notifications pertaining to dataset1 is `http://example.com/dataset1/change/`.

```
POST /websub/ HTTP/1.1
Host: http://hub.example.org
Content-Type: application/xml
Link: <http://example.com/dataset1/change/> ; rel="self",
 <http://hub.example.org/websub/> ; rel="hub",
 <http://www.example.com/dataset1/capabilitylist.xml> ; rel="resourcesync"
Content-Length: 849

<?xml version="1.0" encoding="UTF-8"?>
<urlset ...
```

Example 2: The HTTP POST used by a Source to submit a change notification payload to its hub

4.2. Destination Subscribes to Hub to Receive Notifications

A Destination subscribes to a Source's topic using the process described in the section "Subscribing and Unsubscribing" of WebSub. The process consists of mandatory subscription request and subscription verification phases:

- In the subscription request phase, shown as "subscription" in Figure 3, the Destination indicates it wants to subscribe to notifications pertaining to a Source's topic URI (channel). In order to do so, a Destination provides a form as the body of an HTTP POST issued against the hub URI. This HTTP POST must have an HTTP `Content-Type` header with a value of `application/x-www-form-urlencoded` and the form contains the following information:
  - The mandatory `hub.callback` parameter that has as value the Destination's callback URI, that is the URI to which the hub should submit the notifications pertaining to the Source's topic URI.
  - The mandatory `hub.mode` parameter that has as value `subscribe`.
  - The mandatory `hub.topic` parameter that has as value the Source's topic URI that the Destination wants to subscribe to.
  - The optional `hub.lease_seconds` parameter that has as value the number of seconds that the Destination desires the subscription to remain active. A Destination that provides a value needs to be aware that it may or may not be honored by the hub.

  The hub must respond to a subscription request with an HTTP "202 Accepted" response code as an indication that the request was received and will be verified.
- In the subscription verification phase, shown as "verification" in Figure 3, the hub verifies whether the Destination genuinely intended to subscribe, and whether the Destination has control over the callback URI it provided. In order to do so, the hub issues an HTTP GET against the Destination's callback URI with the following query string arguments appended to it:
  - The mandatory `hub.mode` parameter that has as value `subscribe`.
  - The mandatory `hub.topic` parameter that has as value the topic URI given in the subscription request.
  - The mandatory `hub.challenge` parameter that has as value a hub-generated random string.
  - The mandatory `hub.lease_seconds` parameter that has as value the number of seconds that the hub will keep the subscription active. This actual subscription period may differ arbitrarily from what the Destination requested. It is recommended that the duration of a subscription granted should not be less than 300 seconds (5 minutes) and should not be more than 2678400 seconds (1 month). Although these suggested limits are somewhat arbitrary, the lower limit is intended to prevent overload by frequent subscription

renewals, whereas the upper limit is chosen to ensure that non-cancelled subscriptions expire within a foreseeable period. In order to maintain a continuous subscription, a Destination must take note of the granted subscription period, and it must issue a new subscription request before the indicated period expires if it wants to keep receiving notifications.

The subscriber confirms its intention to subscribe by sending a response that has an HTTP 2XX status code and that has the value of the `hub.challenge` as its body. Any other response indicates that there was no intent to subscribe. .

Example 3 shows the HTTP POST issued by a Destination against the hub URI `http://hub.example.org/websub/` requesting a subscription to the Source's topic URI (channel) `http://example.com/dataset1/change/` as a means to receive change notifications pertaining to dataset1 at its callback URI `http://destination.example.net/callback/`.

```
POST /websub/ HTTP/1.1
Host: http://hub.example.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 141

hub.mode=subscribe&hub.topic=http%3A%2F%2FAexample.com%2Fdataset1%2Fchange%2F
&hub.callback=http%3A%2F%2Fdestination.example.net%2Fcallback%2F&hub.lease_seconds=3600
```

Example 3: A Destination's request to a hub to subscribe to a Source's notification channel

Example 4 shows the HTTP GET issued by the hub against the Destination's callback URI to verify that it was the Destination's intent to subscribe.

```
GET /callback/?hub.mode=subscribe&hub.topic=http%3A%2F%2FAexample.com%2Fdataset1%2Fchange%2F
&hub.challenge=c0cc4630-5116-11e3-8f96-0800200c9a66&hub.lease_seconds=2400 HTTP/1.1
Host: http://destination.example.net
Connection: Close
```

Example 4: A hub's request to verify a Destination's intent to subscribe

Example 5 shows the response by a Destination to the hub's subscription verification request of Example 4. It indicates that the Destination wants the subscription.

```
HTTP/1.1 200 OK
Date: Tue, 19 Nov 2013 12:42::13 GMT
Content-Type: text/plain; charset=UTF-8
Content-Length: 36
Connection: Close

c0cc4630-5116-11e3-8f96-0800200c9a66
```

Example 5: A hub's request to verify a Destination's intent to subscribe

4.3. Hub Delivers Notifications to Destination

When the hub receives a change notification from the Source, it passes it on to the subscribing Destination(s). The process, shown as "Hub notifies Destination" in Figure 3, is as follows:

- The hub provides the notification payload received from the Source as the body of an HTTP POST issued against the Destination's callback URI. This HTTP POST must have an HTTP `Content-Type` header with a value of `application/xml` and an HTTP Link header with the following links:
  - A mandatory link with the `self` relation type that provides the topic URI (channel) for the notification as the value of the `href` attribute.
  - A mandatory link with the `hub` relation type that provides the hub URI as the value of the `href` attribute.
- The successful response from the Destination's callback URI must be have an HTTP 2XX code. The hub must consider all other subscriber response codes as failures.
- The Destination may want to verify the payload received from the Hub, for example, confirming that it is valid XML, has a `<urlset>` root element, etc.

Example 6 shows the HTTP POST that the hub issues against the Destination's callback URI to relay the notification it received from the Source in Example 2. For brevity, the payload is not shown in its entirety.

```
POST /callback/ HTTP/1.1
Host: http://destination.example.net
Content-Type: application/xml
Link: <http://example.com/dataset1/change/> ; rel="self",
 <http://hub.example.org/websub/> ; rel="hub",
 <http://www.example.com/dataset1/capabilitylist.xml> ; rel="resourcesync"
Content-Length: 849

<?xml version="1.0" encoding="UTF-8"?>
<urlset ...
```

Example 6: The HTTP POST used by a hub to submit a Source's change notification payload to a Destination

4.4. Destination Unsubscribes from Hub

The mechanism by which a Destination unsubscribes from a Source's topic URI is as described in Section 4.1 but uses `unsubscribe` as the value of `hub.mode` instead of `subscribe`.

5. Advertising Change Notification Channels

Change Notification capabilities are advertised via Capability Lists, as is the case with the capabilities defined in the core ResourceSync specification. As each set of resources has its dedicated Change Notification channel, that channel is advertised in the Capability List that corresponds with the respective set of resources.

Figure 4 displays a Change Notification channel advertised in a Capability List. The figure shows a structure with only one Capability List that advertises its designated Change Notification channel. Other Capability Lists, each of which pertain to a different set of resources, would advertise their respective notification channels. In addition to Change Notifications, the Capability List can advertise other capabilities such as a Resource List and Change List as introduced in the core specification, and archive capabilities as introduced in the archiving specification.

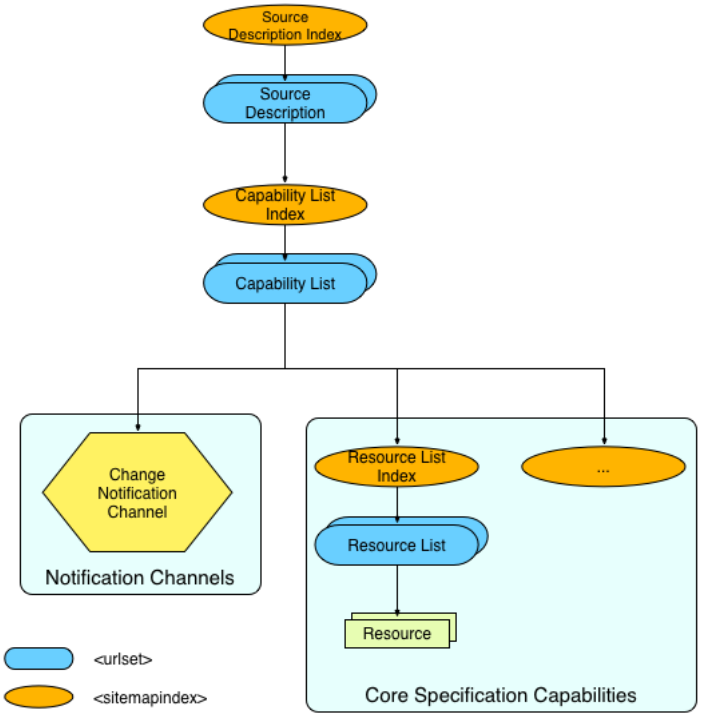

Figure 4: Change Notification channel discovery

Example 7 shows the Capability List from Example 13 of the core specification with discovery links for a Change Notification channel added. The WebSub topic URI is provided in the `<loc>` element, whereas the hub URI is provided using a `<rs:ln>` child element of `<loc>`. That `<rs:ln>` must have `hub` as the value of the `rel` attribute and the hub URI as the value of the `href` attribute. Note the introduction of the `change-notification` value for the `capability` attribute to indicate the Change Notification capability.

```
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
        xmlns:rs="http://www.openarchives.org/rs/terms/">
   <rs:ln rel="describedby"
          href="http://example.com/info_about_set1_of_resources.xml"/>
   <rs:ln rel="up"
          href="http://example.com/source_description.xml"/>
   <rs:md capability="capabilitylist"/>
   <url>
       <loc>http://example.com/dataset1/resourcelist.xml</loc>
       <rs:md capability="resourcelist"/>
   </url>
   <url>
       <loc>http://example.com/dataset1/resourcedump.xml</loc>
       <rs:md capability="resourcedump"/>
   </url>
   <url>
       <loc>http://example.com/dataset1/changelist.xml</loc>
       <rs:md capability="changelist"/>
   </url>
   <url>
       <loc>http://example.com/dataset1/changedump.xml</loc>
       <rs:md capability="changedump"/>
   </url>
   <url>
      <loc>http://example.com/dataset1/change/</loc>
      <rs:ln rel="hub" href="http://hub.example.org/websub/"/>
      <rs:md capability="change-notification"/>
   </url>
</urlset>
```

Example 7: A Capability List with an entry to discover a WebSub change notification channel

6. References

[WebSub]
Genestoux, Julien and Aaron Parecki, eds. *WebSub*, W3C Candidate Recommendation, April 11, 2017.
[Sitemaps]

Sitemaps XML Format. sitemaps.org, last updated February 27, 2008. Available at: http://www.sitemaps.org/protocol.html

[Web Architecture]
Jacobs, Ian, and Norman Walsh, eds. *Architecture of the World Wide Web, Volume One*. W3C Recommendation. World Wide Web Consortium, December 15, 2004. Available at: http://www.w3.org/TR/webarch/

A. Acknowledgements

This specification is the collaborative work of NISO and the Open Archives Initiative. Initial funding for ResourceSync was provided by the Alfred P. Sloan Foundation. UK participation was supported by Jisc.

B. Change Log

Date	Editor	Description
2017-07-20	simeon, martin, herbert	Update to use WebSub instead of PubSubHubbub, consistently use `change-notification` capability value
2017-01-18	herbert, simeon	link to Internet Archive copy of PubSubHubbub, no change to content
2016-08-10	herbert, martin, simeon	version 1.0, removed Framework Notification from the spec, made updates related to Core Framework changes
2014-03-24	graham, herbert	version 0.9, removed ResourceSync-specific requirements from communication between Source and hub
2013-12-18	herbert, martin, rob, simeon	version 0.8.1, using PubSubHubbub
2013-11-12	martin, herbert, rob, simeon	version 0.8, using WebSockets