AddId!

Specification

Version 0.1d, January 2007

Written by Lukas Rosenstock

Introduction and Purpose of AddId!

The idea behind AddId! is to allow websites, which provide useful data that their users might want to use on another site, to provide these users with a very simple method to transfer their data to this other site, usually through the browser.
Just as one example, many blogs today show their visitors a whole bunch of buttons from well-known web-based RSS readers so that the visitors may subscribe to the feed with a simple click. They are, however, useless, if the user uses another service that the writer of the blog might not even know about. With AddId!, the blogging software would find out automatically which feed reader (if any) his reader uses and create the personalized button. There are many other use cases. Sites could offer their users event announcements that can be added to their online calendar with just one click, or contact data for their web-based email address book.

Terminology

There are four parties acting in AddId!: data provider, data collector, identity provider and user.
Data Provider (DP): a website that has some interesting data to offer and wants to give their users a one-click-transfer or -subscription option.
Data Collector (DC): the website that handles and collects the data offered by the provider.
Identity Provider (IdP): in this specification always refers to the party that is responsible for the delivery of the XRDS document.

Data Transfer Methods

There are two methods how a DP can send his data to the DC. The usual way is *browser-based transfer*. For this transfer method, the DP simply constructs a URL based on the URL of the DC with his data (or a reference) in the query. This URL is presented as a link or button on the site. When the user clicks on it, his browser sends a GET request to the DC. The DC will most likely return a page where the user may confirm that he wants this data to be added. The second method is *background transfer*. When using this method, the DP sends a direct POST request to the DC. As there is no possibility for the user to give his consent (unless there is some kind of authorization between DP and DC), the DC should store the new data temporaryly and ask the user later for consent (later is not defined here). Background transfer will be defined in more detail in later versions of AddId!.
Independent from the transfer method, there are again two encoding possibilities: Either the complete dataset is added to the request in Base64-encoding or only a URL is transmitted, that the DC has to fetch. The encoding method depends on the data type.

Special Use Cases

The browser (or a plugin) may parse a web page for information in the metadata or in

microformats and then act as a DP. The link is then shown in the browser menu or toolbar, but apart from this this use case is no different than site-driven browser-based transfer.
On the other hand, local applications may take the role of a DC instead of a web-based service. These are identified with special URI schemes like *myapp:*. For the IdP or DP this means no change. However, they MUST NOT check whether the given URI is a HTTP URL.

Taking AddId! to life

On the IdP

The support for AddId! is included in the XRDS file by adding new services:

```
<Service xmlns:ai="http://addid.identity20.eu/ns">
 <Type>http://addid.identity20.eu/0.1d</Type>
 <ai:supports>ServiceInstaller vCardUpload</ai:supports>
 <URI>http://user.example.org/addid</URI>
</Service>
```

The <Type>-Identifier MUST be `http://addid.identity20.eu/0.1d` for version 0.1d of this spec. The URI is the endpoint given by the DC. The namespace `http://addid.identity20.eu/ns` MUST be added and it is RECOMMENDED to call it *ai*. An <ai:supports> tag with at least one valid AddId! data type MUST be included. It may contain more than one data type, if they are all accepted by the same endpoint. In this case, multiple data types are seperated with whitespaces.
It is RECOMMENDED that, if one DC endpoint supports multiple AddId! data types, that they are all included in one Service definition, in order to keep the document as small as possible.
Priorities MAY be defined, however they are only taken into account if there are multiple services with the same AddId! data types.

On the DC

Create an endpoint at any URL that knows how to handle the parameters that are sent by the request for your supported AddId! data types. The GET request for browser-based transfer MUST NOT actually add something to your database, you MUST show a confirmation message to the user. If a *redirectBackTo* parameter was included in the original request, you MUST send the user back to this URL after the given task is done. If not, for example if the browser acted as DP or the site opened a new window or tab, you MAY include a button for the script function *window.close()*.

If you want to support backgound transfer, whenever the POST request has been acceptable, you should save it and return 202 (accepted) and nothing else to the DP. If you do not want to support background transfer, any POST request to your URL MUST return the HTTP status code 405 (method not allowed).

On the DP

A DP implements the following steps for browser-based transfer:

1. Ask the user for his Yadis URL or XRI/iName. If you do authenticate the user using OpenID, this step is done anyway.
2. Parse the XRDS for all Service definitions with `http://addid.identity20.eu/0.1d` as <Type>-identifier and find those with your desired AddId! data type.
3. If there are multiple, you MUST do the following steps at least for the service or endpoint with the highest priority. You MAY ignore services or endpoints with lower priority.
4. Take the <URI>-value from the service description and add these query parameters (values are URL-encoded, of course):
    1. *userID* is the canonicalized identifier of the user (his Yadis URL or XRI)
    2. *redirectBackTo* is an optional URL of a page that the user should see after the DC has processed the data
    3. *dataType* is the AddId! service type
    4. either *data64* or *url*, according to the AddId! data type

    5. if the data type requires it, additional parameters starting with *x-*
5. Create a link or button with a useful caption and deliver it on your site.

A DP implements the following steps for background transfer:

1. Ask the user for his Yadis URL or XRI/iName. If you do authenticate the user using OpenID, this step is done anyway.
2. Parse the XRDS for all Service definitions with `http://addid.identity20.eu/0.1d` as <Type>-identifier and find those with your desired AddId! data type.
3. If there are multiple, you MUST use the service or endpoint with the highest priority. You MAY ignore services or endpoints with lower priority.
4. Create a list of query-value pairs:
    1. *userID* is the canonicalized identifier of the user (his Yadis URL or XRI)
    2. *redirectBackTo* is an optional URL of a page that the user should see after the DC has processed the data
    3. *dataType* is the AddId! service type
    4. either *data64* or *url*, according to the AddId! data type
    5. if the data type requires it, additional parameters starting with *x-*
5. Send a POST request to the <URI>-value with Content-Type set to application/x-www-form-urlencoded.
6. If 2xx is returned, you're done. If 4xx is returned and similar services with lower priorities exist, you MAY try these.

AddId! data types

You may only use the data types included in the official specification of the used AddId! version. Suggestions for new AddId! data types are welcome!

Basic

ServiceInstaller

Add new Yadis/XRI Services to an existing XRDS document. For example, IdPs might provide a DC for the AddId! ServiceInstaller in order for other DCs to register at the IdP. Uses **data64**. The data is an XML snippet containing the new service description, from (including) <Service> to </Service>.

Bookmark

Add the URL of a website to any service, which could be for example either a private online bookmark storage or a social bookmarking service. Uses **url**.

Content Syndication

AddRSS

Add any newsfeed (e.g. blog or podcast) to a feedreader. Uses **url**. The URL must point to a valid RSS 2.0 feed file.

AddAtom

Add any newsfeed (e.g. blog or podcast) to a feedreader. Uses **url**. The URL must point to a valid ATOM feed file.

Social Networking

AddFriend

Add the Yadis URL or XRI of a friend as a friend on your social network. Uses **url**. The URL must start with http:, https: or xri: and must point, if executed from a browser, to the personal profile of this person.

Personal Information Management (PIM)

vCardUpload

Send data about a contact to an address book service. Uses **data64**. The data is a standard vCard file (classic format, no XML representation).

vCalUpload

Send data about appointments or tasks to a calendar service. Uses **data64**. The data is a standard vCal file (classic format, no XML representation).