ProfileConnect
Gateway API Specification

Proposal
Authored by Jacob Jay
Nov. '08, r02

A lightweight service discovery protocol and programmatic interface for web-services handling a user's personal media assets and between third-parties.

'Profile Connect' aims to simplify and standardised the ability to transfer social media and access profiles—across different providers through an open specification for interoperability.

With such an approach, users benefit from a uniform way of adding accounts for diverse services, whilst both provider and client developers gain wider support amongst each other's applications.

Contents

Model

At the top is the provider which represents the server platform providing methods, to which you as the developer of a consumer application, send requests (calls) to carry out actions on resources of various classes that the provider supports. These requests return results in a format that represent a resource's properties.

A method is the combination of a resource and an action, requested by the consumer using an appropriate protocol. A provider may offer many such resources, methods, and actions, but should support at least one protocol (e.g. HTTP POST) and format (e.g. XML).

Most resources are associated with a user represented as a profile, on whose behalf a consumer interacts with the provider. The provider employs authorization requiring that users grant permission for a consumer to use methods on their behalf, and

authentication requiring that a consumer must both declare that it is authorised to access methods on behalf of a user, as well as to confirm its own identity.

A service represents a process that can be applied to a resource (object) by dispatching it to that service. For example a simple service might an image filter, whilst a gateway provider might have services offering interfaces to remote third-party providers—thus providing broadcast functionality in which a consumer could specify objects to be dispatched (uploaded) to a user's accounts on their authorized third-party services.

Resources

Resources are usually at the core of a provider and represent a variety of elements. All resources are composed of any number of properties that are **settable** as (HTTP) parameters in method **requests**, and **gettable** as (XML) properties from method **responses**.

Properties are detailed for each resource below, but note that when interacting with objects and services, the specific properties vary according to type. Properties are simple name and value pairs or type, name and value **triplets**.

Provider

Represents the server platform. No methods defined at present.

Profiles

Represent users.

Standard Properties

- name
  The user's real name (if any).

Service-types

These may represents a remote provider such as Flickr or Blogger, identified with reverse-canonical domain names (e.g. com.flickr or com.google.blogger). Note that with Redirect Authorization of services a consumer need not ever interact with the service-type object.

Standard Properties

- objects
  - A list of the object types supported, see objects.
- authorization
  - service-specifics
- favicon
  - A URL for an image (GIF).
- logo
  - A URL for an image (PNG).

Services

In the case of a gateway these represent a user's individual third-party accounts such as photo sharing and social networking sites that accept media uploads. A profile may be authorized for multiple services.

Services currently have no standard properties. See the services page for details of adding services, and their respective properties.

Objects

Represent file-based media such as photos and video, or property-based items such as blog posts and presence (status) updates. These can be dispatched (uploaded) to services. Objects have differing properties according to type (see types below), but the following standard properties are supported for most types.

Properties

- types
- title
- description
- keywords
- See services for non-standard properties

Types

Objects behave differently when dispatched to services, and services may only accept objects of specific types or with specific properties. An object may be of multiple (non-conflicting) types simultaneously (type value is a list), but if not specified upon creation a single default type should be assigned by the provider. Types have the following property requirements.

- **file** (default if data is present)
  - All standard properties are optional and valid
  - file:data is required as content, file:name is optional
  - title should not contain the file name
- **article** (default if title and description are present)
-
  - description is required as content, title and keywords are optional
- **presence** (default if only title is present)
  - title or description are valid as content (i.e. the user-defined status)
- **talk** (default if only description is present)
  - title or description are valid as the content (e.g. a microblog post)
  - keywords are valid, and may be appended (e.g. as hashtags)

Transmogrification

If an object cannot be sent to a service a provider may transmogrify it. For example an image could be sent to a microblog service by hosting it and sending a link instead, and to blog services by hosting and referencing in a blog post. Additional methods may be required to facilitate and control this process.

Dispatches

These represent job tickets assigned for processing objects with services. With queued processing a consumer must query retroactively if it wishes to retrieve service-specific response properties after an object is sent.

Properties

- dispatch:created
- See services for non-standard properties

Statuses

- 0 = delivered (confirmed)
- 1 = sent (unconfirmed)
- 2 = failed
- 3 = sending
- 4 = queued
- 5 = preprocessing
- 6 = receiving
- 8 = object not supported
- 9 = not available or unknown

Mechanisms

Protocols

HTTP Request

Should accept requests using HTTP with parameters encoded according to the following Content-Types.

- GET with urlencoded query string arguments
- POST with Content-Type: multipart/x-www-form-urlencoded
- POST with Content-Type: multipart/form-data

HTTP Response

HTTP status codes should only be respected if it is not possible to parse the response.

Response Formats

Content-Type: text/xml

```
<?xml version="1.0"?>
<api provider="com.example" title="Example" version="1.4">
        <response method="create.user" status="0">
         <!-- method response content -->
        </response>
</api>
```

The root element is api (its attributes can generally be ignored) which nests a response element. If you know the method you called, you need not check the method attribute value, however you must check the status attribute value (see statuses). The response element contains the specific elements and properties from the requested method.

Response Statuses

All requests return a response status attribute in the response element with the value corresponding to a status code number. A non-zero number indicates a failed request, if the value is less than 100 it corresponds to the standard errors below, but if it is greater than 100 it indicates a method specific failure code.

- 0 = success

- 1 = authentication required
- 2 = authentication failed
- 6 = temporary fault
- 20 = required parameters missing

The response element may include one or more message elements that should be logged or displayed to a user. With sucessful requests (status 0) these represent method-specific warnings, and with failed requests (status > 0) will further describe the error causing failure. The message element's contents are always a full description, and it may have an optional **title**, or **url** attribute.

```
  <message title="Invalid Authentication" url="">No valid credentials were
```

Value Notations

Text

Should be UTF-8 encoded. Linebreaks should be UNIX-style using the LF character (ASCII 10) only.

Serialised Lists

A list may contain one or more text values or serialised properties. Multiple values must be delimited with the semi-colon character (ASCII 59) and without white-space. A list should not be terminated with the delimiter, and qualification of values containing spaces or other special values is not required. The ;= (semi-colon, equals) characters are illegal in values and must be removed, replaced or percent-encoded.

- **Serialised Properties**

A property is a name-value pair where the format is name=value delimited with an =

(equals) character (e.g. "wibble=wobble" specifies a name of "wibble" and a value of "wobble"). The ;:= (semi-colon, colon, equals) characters are illegal in names and values and must be removed, replaced or their values should be percent-encoded.

Methods

profile.create

Note: This method is for use only when Web Authorization is impractical.

Create a new profile using the properties specified by the consumer making the request, and authorise it to access the profile by returning a Profile Token.

The token should be saved securely (preferably encrypted) by the consumer application, and is to be used when requesting resources requiring authentication with a profile token.

Request

Authentication

- Consumer
- Provider [au]

Parameters

- method
  (Required) connect.profile.create
- login *(Optional)*
  User's email address.
- pass *(Optional)*
  User's password if a login is specified.
- name *(Optional)*
  User's first or full name.

Warning: If no login is specified the user can only be logged in using Redirect Access and should the issued Profile Token become lost the account will be unrecoverable. We recommend specifying a login where possible.

Response

XML

```
<resource class="profile" id="[id]" name="[name]">
 <property type="auth" name="token">[token]</property>
</resource>
```

Statuses

- Standard

profile.consumer.authorize

Note: This method is for use when Web Authorization is impractical.

From the profile (user) credentials specified by the consumer making the request, authorise it to access the profile by returning a Profile Token.

The token should be saved securely (preferably encrypted) by the consumer application, and is to be used when requesting resources requiring authentication with a profile token. A consumer must never save or cache the user's Applications Key.

Request

Authentication

- Consumer
- Provider [ac]

Parameters

- method
  (Required) connect.profile.consumer.authorize
- login *(Required)*
  Value is user's registered *email address*.
- pass *(Required)*
  Value is user's *Applications Key* (not password) available to the user from their settings on the site.

Response

XML

```
<resource class="profile" id="[id]" name="[name]">
 <property type="auth" name="token">[token]</property>
</resource>
```

Statuses

- Standard

profile.service.subscribe

Subscribe a new service to a profile using the specified credentials if required. See services for more details on working with services and their responses.

Request

Authentication

- Profile
- Consumer

Parameters

- method *(Required)*
  connect.profile.service.subscribe

- service-type *(Required)*
  A service-type (from provider.services.fetch).
- login *(Conditional)*
- pass *(Conditional)*
- url *(Conditional)*

Condition: Check the properties returned from provider.services.fetch for the desired service type as different services may require different parameters and also may specify parameter field labels to be displayed to the user when asking for input to avoid confusion over values (e.g. login may require 'email address' or 'user name').

Response

XML

```
<resource class="service" id="[id]" name="[name]">
 <!-- if called with a valid credentials -->
 <property type="[type]" name="[name]">[value]</property>
 <!-- if called without valid credentials -->
 <property type="authorization" name="types">web;parameters;none</property>
 <!-- if authorization:types=web -->
 <property type="authorization" name="url">http://[…]</property>
 <!-- if authorization:types=parameters -->
 <property type="parameter" name="login">Email Address</property>
 <property type="parameter" name="pass">Password</property>
 <property type="parameter" name="url">Blog URL</property>
</resource>
```

Statuses

- 101 = Unknown service type
- Standard

profile.services.fetch

Get the services associated with a profile. This allows a consumer to find out which service types (from provider.services.fetch) a user is subscribed to.

Request

Authentication

- Profile
- Consumer

Parameters

- method *(Required)*
  connect.profile.services.fetch

Response

XML

```
<service id="0" type="default" name="My Default Pipes" />
```

```
<service id="[id]" type="[type]" name="[name]">
 <property type="" name="default">[value]</property>
</service>
```

Note: Response always includes a default service with an id of 0, which represents a user's enabled (default) destinations, dispatches sent to this service will actually send to all enabled services. Consumers may wish to filter it out when dispalying services to users.

Statuses

- Standard codes

object.create

Create a new object resource (e.g. an image or presence update) on behalf of a user, with the specified properties, and recieve an object ID. *May also be used to dispatch the object to services without requiring an additional request to the objects.dispatch method.*

Request

Parameters

- method *(Required)*
  connect.object.create
- types (*Conditional*)
  A semi-colon delimited list of types, if not specified we will automatically ascertain the type.
- title *(Conditional)*
  A concise summary of the object, i.e. a name. (For a status update this would be the status string.)
- description *(Conditional)*
  Text elaborating the content of the object. (For a blog post this would be the article content.)
- data *(Conditional)*
  A file to upload such as an image or video. Presence of this value requires Content-Type: **multipart/form-data**.
- keywords *(Optional)*
  A semi-colon delimited list of words ('tags') summarising the content of the object.
- service-ids *(Optional)*
  A list of service IDs to which the object will be dispatched immediately. If this is not specified the object will not be dispatched to any services, and the objects.dispatch method should be called.
- property:*
  *(Optional)*Parameters prefixed with *property:* can be specified and will be created as additional object properties. This enables handling of service-specify properties. These properties are also returned by objects.fetch.

Condition: At least one of either type, title, description or file must be present or the object will not be created.

Authentication

- Profile
- Consumer

Response

XML

```
<resource class="object" id="[id]" uri="" />
<resource class="dispatch" id="[id]" service="[id]" status="5" />
```

Status

- Standard codes

Messages

- Too many properties. Only the first 25 created.
- Property too long. Properties whose name and value exceeded 128 characters were discarded.

profile.objects.fetch

Get the properties of an object. Currently this serves no useful purpose other than to retrieve properties set during object creation if the consumer has not stored them locally.

Request

Authentication

- Profile
- Consumer

Parameters

- method *(Required)*
  connect.profile.objects.fetch
- object-ids *(Required)*
  One or more semi-colon delimited object IDs.

Response

XML

```
<object id="[id]">
 <property type="[type]" name="[name]">[value]</property>
</object>
```

Statuses

- Standard

objects.dispatches.fetch

For the specified object IDs, get the processing status and service response properties for **all** services that each object has been dispatched to.

Request

Authentication

- Profile
- Consumer

Parameters

- method *(Required)*
  connect.objects.dispatches.fetch
- object-ids *(Required)*
  One or more semi-colon delimited object IDs.

Response

XML

```
<resource class="dispatch" id="[id]" status="0" object="[id]" service="[id]
 <property type="dispatch" name="created">[timestamp]</property>
 <property type="[type]" name="[name]">[value]</property>
</resource>
```

Note: Resource element may be empty. Multiple resource elements may be returned.

Statuses

- 103 = Too many object IDs specified, max 25.
- Standard codes

profile.objects.dispatch

For a specified object dispatch it to the specified service and get a dispatch ID. A consumer may then use the ID to make a retro-active request to dispatches.fetch and get the service response. *Note that you can also dispatch objects with object.create.*

Request

Authentication

- Profile
- Consumer

Parameters

- method *(Required)*
  connect.profile.objects.dispatch
- object-ids *(Required)*
  One or more semi-colon delimited object IDs.
- service-ids *(Required)*
  One or more semi-colon delimited service IDs (from profile.services.fetch). To send to a user's default destinations use service ID 0.

Note: All specified objects will be sent to all matching services. We automatically match objects to the types supported by a service and do not dispatch if they do not match or cannot be transmogrified.

Response

XML

```
<resource class="dispatch" id="[id]" object="[id]" service="[id]" status="5
```

Note: Multiple resource elements may be returned.

Statuses

- 103 = Too many object-ids or service-ids specified, max 100.
- Standard codes

profile.objects.dispatch

For a specified object dispatch it to the specified service and get a dispatch ID. A consumer may then use the ID to make a retro-active request to dispatches.fetch and get the service response. *Note that you can also dispatch objects with object.create.*

Request

Authentication

- Profile
- Consumer

Parameters

- method *(Required)*
  connect.profile.objects.dispatch
- object-ids *(Required)*
  One or more semi-colon delimited object IDs.
- service-ids *(Required)*
  One or more semi-colon delimited service IDs (from profile.services.fetch). To send to a user's default destinations use service ID 0.

Note: All specified objects will be sent to all matching services. We automatically match objects to the types supported by a service and do not dispatch if they do not match or cannot be transmogrified.

Response

XML

```
<resource class="dispatch" id="[id]" object="[id]" service="[id]" status="5
```

Note: Multiple resource elements may be returned.

Statuses

- 103 = Too many object-ids or service-ids specified, max 100.
- Standard codes

dispatches.fetch

For the specified dispatch IDs, get the processing status and service response properties. *To get all dispatches for an object use objects.dispatches.fetch.*

Request

Authentication

- Profile
- Consumer

Parameters

- method *(Required)*
  connect.dispatches.fetch
- ids *(Required)*
  One or more semi-colon delimited dispatch IDs.

Response

XML

```
<resource class="dispatch" id="[id]" status="0" object="[id]" service="[id]
 <property type="dispatch" name="created">[timestamp]</property>
 <property type="[type]" name="[name]">[value]</property>
</resource>
```

Note: Resource element may be empty. Multiple resource elements will be returned if multiple dispatch IDs are specified.

Statuses

- 103 = Too many dispatch IDs specified, max 100.
- Standard codes