

Spring Social Yammer Reference Manual

Morten Andersen-Gott

Spring Social Yammer Reference Manual

by Morten Andersen-Gott

1.0.0.M3

© SpringSource Inc., 2011

Table of Contents

1. Spring Social Yammer Overview	1
1.1. Introduction	1
1.2. How to get	1
2. Configuring Yammer Connectivity	2
3. Yammer API Binding	4
3.1. Retrieving a user's Yammer profile data	5
3.2. Listing Yammer users in a network	5
3.3. Get messages posted to your network	6
3.4. Post an update to your network	6

1. Spring Social Yammer Overview

1.1 Introduction

The Spring Social Yammer project is an extension to [Spring Social](#) that enables integration with Yammer.

With 3 million users (and growing) from 80 percent of the Fortune 500 companies, [Yammer](#) is major platform for enterprise social networks.

Spring Social Yammer enables integration with Yammer with `YammerConnectionFactory`, a connection factory that can be plugged into Spring Social's service provider connection framework, and with an API binding to Yammer's REST API.

1.2 How to get

The following Maven dependency will add Spring Social Yammer to your project:

```
<dependency>
  <groupId>org.springframework.social</groupId>
  <artifactId>spring-social-yammer</artifactId>
  <version>${org.springframework.social-yammer-version}</version>
</dependency>
```

As an extension to Spring Social, Spring Social Yammer depends on Spring Social. Spring Social's core module will be transitively resolved from the Spring Social Yammer dependency. If you'll be using Spring Social's web module, you'll need to add that dependency yourself:

```
<dependency>
  <groupId>org.springframework.social</groupId>
  <artifactId>spring-social-web</artifactId>
  <version>${org.springframework.social-version}</version>
</dependency>
```

Note that Spring Social Yammer may release on a different schedule than Spring Social. Consequently, Spring Social's version may differ from that of Spring Social Yammer.

Consult [Spring Social's reference documentation](#) for more information on Spring Social dependencies.

2. Configuring Yammer Connectivity

Spring Social's `ConnectController` works with one or more provider-specific `ConnectionFactory`s to exchange authorization details with the provider and to create connections. Spring Social Yammer provides `YammerConnectionFactory`, a `ConnectionFactory` for creating connections with Yammer.

So that `ConnectController` can find `YammerConnectionFactory`, it must be registered with a `ConnectionFactoryRegistry`. The following class constructs a `ConnectionFactoryRegistry` containing a `ConnectionFactory` for Yammer using Spring's Java configuration style:

```
@Configuration
public class SocialConfig {

    @Bean
    public ConnectionFactoryLocator connectionFactoryLocator() {
        ConnectionFactoryRegistry registry = new ConnectionFactoryRegistry();
        registry.addConnectionFactory(new YammerConnectionFactory(
            environment.getProperty("yammer.clientId"),
            environment.getProperty("yammer.clientSecret")));
        return registry;
    }
}
```

Here, a Yammer connection factory is registered with `ConnectionFactoryRegistry` via the `addConnectionFactory()` method. If we wanted to add support for connecting to other providers, we would simply register their connection factories here in the same way as `YammerConnectionFactory`.

Because client IDs and secrets may be different across environments (e.g., test, production, etc) it is recommended that these values be externalized. As shown here, Spring 3.1's `Environment` is used to look up the application's client ID and secret.

Optionally, you may also configure `ConnectionFactoryRegistry` and `YammerConnectionFactory` in XML:

```
<bean id="connectionFactoryLocator" class="org.springframework.social.connect.support.ConnectionFactoryRegistry">
    <property name="connectionFactories">
        <list>
            <bean class="org.springframework.social.yammer.connect.YammerConnectionFactory">
                <constructor-arg value="${yammer.clientId}" />
                <constructor-arg value="${yammer.clientSecret}" />
            </bean>
        </list>
    </property>
</bean>
```

This is functionally equivalent to the Java-based configuration of `ConnectionFactoryRegistry` shown before. The only casual difference is that the connection factories are injected as a list into the

`connectionFactories` property rather than with the `addConnectionFactory()` method. As in the Java-based configuration, the application's client ID and secret are externalized (shown here as property placeholders).

Refer to [Spring Social's reference documentation](#) for complete details on configuring `ConnectController` and its dependencies.

3. Yammer API Binding

Spring Social Yammer offers integration with Yammer's REST API with the `Yammer` interface and its implementation, `YammerTemplate`.

Spring Social Yammer's `Yammer` interface and its implementation, `YammerTemplate` provide the operations needed to interact with Yammer on behalf of a user. Creating an instance of `YammerTemplate` is as simple as constructing it by passing in an authorized access token to the constructor:

```
String accessToken = "f8FX29g..."; // access token received from Yammer after OAuth authorization
Yammer yammer = new YammerTemplate(accessToken);
```

If you are using Spring Social's [service provider framework](#), you can get an instance of `Yammer` from a `Connection`. For example, the following snippet calls `getApi()` on a connection to retrieve a `Yammer`:

```
Connection<Yammer> connection = connectionRepository.findPrimaryConnection(Yammer.class);
if (connection != null) {
    Yammer yammer = connection.getApi();

    // ... use Yammer API binding
}
```

Here, `ConnectionRepository` is being asked for the primary connection that the current user has with Yammer. If a connection to Yammer is found, it retrieves a `Yammer` instance that is configured with the connection details received when the connection was first established.

Once you have a `Yammer` you can use it to interact with Yammer on behalf of the user who the access token was granted for.

With a `Yammer` in hand, there are several ways you can use it to interact with Yammer on behalf of the user. Spring Social's Yammer API binding is divided into 7 sub-APIs exposed through the methods of `Yammer`:

```
public interface Yammer {

    ThreadOperations threadOperations();

    SubscriptionOperations subscriptionOperations();

    TopicOperations topicOperations();

    SearchOperations searchOperations();

    GroupOperations groupOperations();

    MessageOperations messageOperations();

}
```

```
UserOperations userOperations();

}
```

The sub-API interfaces returned from Yammer's methods are described in Table 3.1, “Yammer's Sub-APIs”.

Table 3.1. Yammer's Sub-APIs

Sub-API Interface	Description
ThreadOperations	Get information about Yammer threads
SubscriptionOperations	Follow, unfollow and check whether you are following topics (tags), users or threads
TopicOperations	Get information about a single Topic (tag)
SearchOperations	Search for messages, users, tags and groups in your Yammer network
GroupOperations	List, get, create, join and leave groups
MessageOperations	Get, post and delete messages
UserOperations	List users, get, and update user.

The following sections will give an overview of common tasks that can be performed via Yammer and its sub-APIs. For complete details on all of the operations available, refer to the JavaDoc.

3.1 Retrieving a user's Yammer profile data

To retrieve the authenticated user's profile data, call the `getUserProfile()` method provided by the `UserOperations`:

```
YammerProfile profile = yammer.userOperations().getUserProfile();
```

The data returned in the `YammerProfile` includes the user's Yammer ID, first and last names, contact information, education and work experience as well as stats such as number of posts, followers and number of people the user is following

3.2 Listing Yammer users in a network

To retrieve a list of users (members) of a Yammer network, call the `getUsers()` method provided by the `UserOperations`:

```
YammerProfile profile = yammer.userOperations().getUsers(1);
```


The single parameter is the page number. User lists are retrieved per page, with Yammer returning a maximum of 50 users per page. The method returns a `List` of `YammerProfiles`.

You may also the overloaded `getUsers()` method that in addition to the page number lets you specify the sorting key (followers or messages), whether you want results in reversed order (true or false, false is default) and the letter you want the users to start with:

```
YammerProfile profile = yammer.userOperations().getUsers(1, UserOperations.SORT_BY_MESSAGES, false, 'A')
```

3.3 Get messages posted to your network

To get messages posted to your network call the `getMessages()` method provided by the `MessageOperations`:

```
YammerProfile profile = yammer.messageOperations().getMessages(0, 0, MessageOperations.NO_THREADING, 0);
```

The parameters to the `getMessages()` are: older than id (returns only messages older than the message ID specified, 0 if you want the latest), newer than id, threading option (`NO_THREADING`, `THREADED` or `THREADED_EXTENDED`), limit (the number of posts per poll, 0 for defaulting to yammer default of 50) The data returned is `MessageInfo` which contains meta data about the poll and a `List` of `YammerMessage`

Additional operations to get messages exists in `MessageOperations`, they all take the same parameters as `getMessages()`. `getMessagesFollowing()` to get messages from people you are following. `getMessagesFromUser()` to get messages posted by a specified user. `getMessagesPrivate()` to private messages. `getMessagesReceived()` to get messages received. `getMessagesLikedByUser()` to get messages liked by a specified user. `getMessagesAboutTopic()` to get messages on a specified topic (topic id).

3.4 Post an update to your network

To post an update to your followers, call the `postUpdate()` method provided by the `MessageOperations`:

```
YammerProfile profile = yammer.messageOperations().postUpdate("Hello all Yammerites!");
```

The data returned is `MessageInfo` which is the same type of object returned when you fetch messages. This allows you to immediately display the newly-posted message without doing a poll.

Optionally, you may provide additional information to your update such as attachments. You may also include a group id if you are posting to a group or a replied to id if you are replying to an existing post. This is done by calling the overloaded `postUpdate()` method provided by the `MessageOperations`:

```
Resource resource = new FileSystemResource("file:///test.txt");
YammerPostDetails details = new YammerPostDetails();
details.addAttachment(resource);
YammerProfile profile = yammer.messageOperations().postUpdate("Hello all Yammerites!", details);
```