Spring Social Facebook Reference Manual

1.1.0.M2

Craig Walls , Keith Donald

Table of Contents

1. Spring Social Facebook Overview

1.1 Introduction

The Spring Social Facebook project is an extension to Spring Social that enables integration with Facebook.

With over 700 million users (and growing), Facebook is the largest online social network. While bringing together friends and family, Facebook also offers a rich platform on which to develop applications.

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

1.2 How to get

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

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

As an extension to Spring Social, Spring Social Facebook depends on Spring Social. Spring Social's core module will be transitively resolved from the Spring Social Facebook 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 Facebook may release on a different schedule than Spring Social. Consequently, Spring Social's version may differ from that of Spring Social Facebook.

Consult Spring Social's reference documentation for more information on Spring Social dependencies.

2. Configuring Facebook Connectivity

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

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

```
@Configuration
public class SocialConfig {

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

}
```

Here, a Facebook 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 FacebookConnectionFactory.

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 FacebookConnectionFactory in XML:

```
<bean id="connectionFactoryLocator" class="org.springframework.social.connect.support.ConnectionFactoryRegi
    <property name="connectionFactories">
        <list>

  <bean class="org.springframework.social.facebook.connect.FacebookConnectionFactory">
                <constructor-arg value="${facebook.clientId}" />
                <constructor-arg value="${facebook.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. Facebook API Binding

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

```
String accessToken = "f8FX29g..."; // access token received from Facebook after OAuth
 authorization
Facebook facebook = new FacebookTemplate(accessToken);
```

In addition, `FacebookTemplate` has a default constructor that creates an instance without any OAuth credentials:

```
Facebook facebook = new FacebookTemplate();
```

When constructed with the default constructor, `FacebookTemplate` will allow a few simple operations that do not require authorization, such as retrieving a specific user's profile. Attempting other operations, such as posting a status update will fail with an `MissingAuthorizationException` being thrown.

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

```
Connection<Facebook> connection =
 connectionRepository.findPrimaryConnection(Facebook.class);
Facebook facebook = connection != null ? connection.getApi() : new FacebookTemplate();
```

Here, `ConnectionRepository` is being asked for the primary connection that the current user has with Facebook. If a connection to Facebook is found, a call to `getApi()` retrieves a `Facebook` instance that is configured with the connection details received when the connection was first established. If there is no connection, a default instance of `FacebookTemplate` is created.

With a `Facebook` in hand, there are several ways you can use it to interact with Facebook on behalf of the user. Spring Social's Facebook API binding is divided into 9 sub-APIs exposes through the methods of `Facebook`:

```
public interface Facebook extends GraphApi {

    CommentOperations commentOperations();

    EventOperations eventOperations();

    FeedOperations feedOperations();

    FriendOperations friendOperations();

    GroupOperations groupOperations();

    LikeOperations likeOperations();

    MediaOperations mediaOperations();

    PlacesOperations placesOperations();

    UserOperations userOperations();

}
```

The sub-API interfaces returned from `Facebook`'s methods are described in Table 3.1, "Facebook's Sub-APIs".

Table 3.1. Facebook's Sub-APIs

Sub-API Interface	Description
CommentOperations	Add, delete, and read comments on Facebook objects.
EventOperations	Create and maintain events and RSVP to event invitations.
FeedOperations	Read and post to a Facebook wall.
FriendOperations	Retrieve a user's friends and maintain friend lists.
GroupOperations	Retrieve group details and members.
LikeOperations	Retrieve a user's interests and likes. Like and unlike objects.
MediaOperations	Maintain albums, photos, and videos.
PlacesOperations	Checkin to location in Facebook Places and retrieve places a user and their friends have checked into.
UserOperations	Retrieve user profile data and profile images.

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

3.1 Retrieving a user's profile data

You can retrieve a user's Facebook profile data using `Facebook`' `getUserProfile()` method:

```
FacebookProfile profile = facebook.userOperations().getUserProfile();
```

The `FacebookProfile` object will contain basic profile information about the authenticating user, including their first and last name and their Facebook ID. Depending on what authorization scope has been granted to the application, it may also include additional details about the user such as their email address, birthday, hometown, and religious and political affiliations. For example, `getBirthday()` will return the current user's birthday if the application has been granted "user_birthday" permission; null otherwise. Consult the JavaDoc for `FacebookProfile` for details on which permissions are required for each property.

If all you need is the user's Facebook ID, you can call `getProfileId()` instead:

```
String profileId = facebook.userOperations().getProfileId();
```

Or if you want the user's Facebook URL, you can call `getProfileUrl()`:

```
String profileUrl = facebook.userOperations().getProfileUrl();
```

3.2 Getting a user's Facebook friends

An essential feature of Facebook and other social networks is creating a network of friends or contacts. You can access the user's list of Facebook friends by calling the `getFriendIds()` method from `FriendOperations`:

```
List<String> friendIds = facebook.friendOperations().getFriendIds();
```

This returns a list of Facebook IDs belonging to the current user's list of friends. This is just a list of `String` IDs, so to retrieve an individual user's profile data, you can turn around and call the `getUserProfile()`, passing in one of those IDs to retrieve the profile data for an individual user:

```
FacebookProfile firstFriend = facebook.userOperations().getUserProfile(friendIds.get(0));
```

Or you can get a list of user's friends as `FacebookProfiles` by calling `getFriendProfiles()`:

```
List<FacebookProfile> friends = facebook.friendOperations().getFriendProfiles();
```

Facebook also enables users to organize their friends into friend lists. To retrieve a list of the authenticating user's friend lists, call `getFriendLists()` with no arguments:

```
List<Reference> friends = facebook.friendOperations().getFriendLists();
```

You can also retrieve a list of friend lists for a specific user by passing the user ID (or an alias) to `getFriendLists()`:

```
List<Reference> friends = facebook.friendOperations().getFriendLists("habuma");
```

`getFriendLists()` returns a list of `Reference` objects that carry the ID and name of each friend list.

To retieve a list of friends who are members of a specific friend list call `getFriendListMembers()`, passing in the ID of the friend list:

```
List<Reference> friends = facebook.friendOperations().getFriendListMembers("193839228");
```

`FriendOperations` also support management of friend lists. For example, the `createFriendList()` method will create a new friend list for the user:

```
Reference collegeFriends = facebook.friendOperations().createFriendList("College
 Buddies");
```

`createFriendList()` returns a `Reference` to the newly created friend list.

To add a friend to the friend list, call `addToFriendList()`:

```
facebook.friendOperations().addToFriendList(collegeFriends.getId(), "527631174");
```

`addToFriendList()` takes two arguments: The ID of the friend list and the ID (or alias) of a friend to add to the list.

In a similar fashion, you may remove a friend from a list by calling `removeFromFriendList()`:

```
facebook.friendOperations().removeFromFriendList(collegeFriends.getId(), "527631174");
```

3.3 Posting to and reading feeds

To post a message to the user's Facebook wall, call `FeedOperations`' `updateStatus()` method, passing in the message to be posted:

```
facebook.feedOperations().updateStatus("I'm trying out Spring Social!");
```

If you'd like to attach a link to the status message, you can do so by passing in a `FacebookLink` object along with the message:

```
FacebookLink link = new FacebookLink("http://www.springsource.org/spring-social",
        "Spring Social",
        "The Spring Social Project",
        "Spring Social is an extension to Spring to enable applications to connect with
 service providers.");
facebook.feedOperations().updateStatus("I'm trying out Spring Social!", link);
```

When constructing the `FacebookLink` object, the first parameter is the link's URL, the second parameter is the name of the link, the third parameter is a caption, and the fourth is a description of the link.

If you want to read posts from a user's feed, `FeedOperations` has several methods to choose from. The `getFeed()` method retrieves recent posts to a user's wall. When called with no parameters, it retrieves posts from the authenticating user's wall:

```
List<Post> feed = facebook.feedOperations().getFeed();
```

Or you can read a specific user's wall by passing their Facebook ID to `getFeed()`:

```
List<Post> feed = facebook.feedOperations().getFeed("habuma");
```

In any event, the `getFeed()` method returns a list of `Post` objects. The `Post` class has six subtypes to represent different kinds of posts:

- `CheckinPost` - Reports a user's checkin in Facebook Places.
- `LinkPost` - Shares a link the user has posted.
- `NotePost` - Publicizes a note that the user has written.
- `PhotoPost` - Announces a photo that the user has uploaded.
- `StatusPost` - A simple status.
- `VideoPost` - Announces a video that the user has uploaded.

The `Post`'s `getType()` method identifies the type of `Post`.