# Spring Social Twitter Reference Manual

1.0.3.BUILD-SNAPSHOT

Craig Walls , Keith Donald

# Table of Contents

# 1. Spring Social Twitter Overview

## 1.1 Introduction

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

Twitter is a popular micro-blogging and social networking service, enabling people to communicate with each other 140 characters at a time.

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

## 1.2 How to get

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

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

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

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

# 2. Configuring Twitter 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 Twitter provides `TwitterConnectionFactory`, a `ConnectionFactory` for creating connections with Twitter.

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

```
@Configuration
public class SocialConfig {

    @Inject
    private Environment environment;

    @Bean
    public ConnectionFactoryLocator connectionFactoryLocator() {
        ConnectionFactoryRegistry registry = new ConnectionFactoryRegistry();
        registry.addConnectionFactory(new TwitterConnectionFactory(
            environment.getProperty("twitter.consumerKey"),
            environment.getProperty("twitter.consumerSecret")));
        return registry;
    }

}
```

Here, a Twitter 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 `TwitterConnectionFactory`.

Because consumer keys 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 consumer key and secret.

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

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

  <bean class="org.springframework.social.twitter.connect.TwitterConnectionFactory">
                <constructor-arg value="${twitter.consumerKey}" />
                <constructor-arg value="${twitter.consumerSecret}" />
            </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 consumer key 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. Twitter API Binding

Spring Social Twitter offers integration with Twitter's REST API through the `Twitter` interface and its implementation, `TwitterTemplate`.

Creating an instance of `TwitterTemplate` involves invoking its constructor, passing in the application's OAuth credentials and an access token/secret pair authorizing the application to act on a user's behalf. For example:

```
String consumerKey = "..."; // The application's consumer key
String consumerSecret = "..."; // The application's consumer secret
String accessToken = "..."; // The access token granted after OAuth authorization
String accessTokenSecret = "..."; // The access token secret granted after OAuth
 authorization
Twitter twitter = new TwitterTemplate(consumerKey, consumerSecret, accessToken,
 accessTokenSecret);
```

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

```
Twitter twitter = new TwitterTemplate();
```

When constructed with the default constructor, `TwitterTemplate` will allow a few simple operations that do not require authorization, such as searching. Attempting other operations, such as tweeting will fail with an `MissingAuthorizationException` being thrown.

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

```
Connection<Twitter> connection =
 connectionRepository.findPrimaryConnection(Twitter.class);
Twitter twitter = connection != null ? connection.getApi() : new TwitterTemplate();
```

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

Once you have a `Twitter`, you can perform a several operations against Twitter. `Twitter` is defined as follows:

```
public interface Twitter {

   boolean isAuthorizedForUser();

   DirectMessageOperations directMessageOperations();

   FriendOperations friendOperations();

   GeoOperations geoOperations();

   ListOperations listOperations();

   SearchOperations searchOperations();

   TimelineOperations timelineOperations();

   UserOperations userOperations();

}
```

The `isAuthorizedForUser` helps determine if the `Twitter` instance has been created with credentials to perform on behalf of a user. It will return true if it is capable of performing operations requiring authorization; false otherwise.

The remaining six methods return sub-APIs, partitioning the Twitter service API into divisions targeting specific facets of Twitter functionality. These sub-APIs are defined by interfaces described in Table 3.1, “Twitter's Sub-APIs”.

*Table 3.1. Twitter's Sub-APIs*

| Sub-API Interface | Description |
|---|---|
| DirectMessageOperations | Reading and sending direct messages. |
| FriendOperations | Retrieving a user's list of friends and followers and following/unfollowing users. |
| GeoOperations | Working with locations. |
| ListOperations | Maintaining, subscribing to, and unsubscribing from user lists |
| SearchOperations | Searching tweets and viewing search trends |
| TimelineOperations | Reading timelines and posting tweets. |
| UserOperations | Retrieving user profile data. |

What follows is a survey of common tasks you may perform with `Twitter` and its sub-APIs. For complete details on the Spring Social's entire Twitter API binding, refer to the JavaDoc.

## 3.1 Retrieving a user's Twitter profile data

To get a user's Twitter profile, call `UserOperations`' `getUserProfile()`:

```
TwitterProfile profile = twitter.userOperations().getUserProfile();
```

This returns a `TwitterProfile` object containing profile data for the authenticated user. This profile information includes the user's Twitter screen name, their name, location, description, and the date that they created their Twitter account. Also included is a URL to their profile image.

If you want to retrieve the user profile for a specific user other than the authenticated user, you can so do by passing the user's screen name as a parameter to `getUserProfile()`:

```
TwitterProfile profile = twitter.userOperations().getUserProfile("habuma");
```

If all you need is the screen name for the authenticating user, then call `UserOperations.getScreenName()`:

```
String profileId = twitter.userOperations().getScreenName();
```

## 3.2 Tweeting

To post a message to Twitter the simplest thing to do is to pass the message to the `updateStatus()` method provided by `TimelineOperations`:

```
twitter.timelineOperations().updateStatus("Spring Social is awesome!")
```

Optionally, you may also include metadata about the tweet, such as the location (latitude and longitude) you are tweeting from. For that, pass in a `StatusDetails` object, setting the location property:

```
StatusDetails statusDetails = new StatusDetails().setLocation(51.502f, -0.126f);
twitter.timelineOperations().updateStatus("I'm tweeting from London!", statusDetails)
```

To have Twitter display the location in a map (on the Twitter web site) then you should also set the `displayCoordinates` property to `true`:

```
StatusDetails statusDetails = new StatusDetails().setLocation(51.502f,
 -0.126f).setDisplayCoordinates(true);
twitter.timelineOperations().updateStatus("I'm tweeting from London!", statusDetails)
```

If you'd like to retweet another tweet (perhaps one found while searching or reading the Twitter timeline), call the `retweet()` method, passing in the ID of the tweet to be retweeted:

```
long tweetId = tweet.getId();
twitter.timelineOperations().retweet(tweetId);
```

Note that Twitter disallows repeated tweets. Attempting to tweet or retweet the same message multiple times will result in a `DuplicateTweetException` being thrown.

## 3.3 Reading Twitter timelines

From a Twitter user's perspective, Twitter organizes tweets into four different timelines:

- User - Includes tweets posted by the user.
- Friends - Includes tweets from the user's timeline and the timeline of anyone that they follow, with the exception of any retweets.
- Home - Includes tweets from the user's timeline and the timeline of anyone that they follow.
- Public - Includes tweets from all Twitter users.

To be clear, the only difference between the home timeline and the friends timeline is that the friends timeline excludes retweets.

`TimelineOperations` also supports reading of tweets from one of the available Twitter timelines. To retrieve the 20 most recent tweets from the public timeline, use the `getPublicTimeline()` method:

```
List<Tweet> tweets = twitter.timelineOperations().getPublicTimeline();
```

`getHomeTimeline()` retrieves the 20 most recent tweets from the user's home timeline:

```
List<Tweet> tweets = twitter.timelineOperations().getHomeTimeline();
```

Similarly, `getFriendsTimeline()` retrieves the 20 most recent tweets from the user's friends timeline:

```
List<Tweet> tweets = twitter.timelineOperations().getFriendsTimeline();
```

To get tweets from the authenticating user's own timeline, call the `getUserTimeline()` method:

```
List<Tweet> tweets = twitter.timelineOperations().getUserTimeline();
```

If you'd like to retrieve the 20 most recent tweets from a specific user's timeline (not necessarily the authenticating user's timeline), pass the user's screen name in as a parameter to `getUserTimeline()`:

```
List<Tweet> tweets = twitter.timelineOperations().getUserTimeline("rclarkson");
```

In addition to the four Twitter timelines, you may also want to get a list of tweets mentioning the user. The `getMentions()` method returns the 20 most recent tweets that mention the authenticating user:

```
List<Tweet> tweets = twitter.timelineOperations().getMentions();
```

## 3.4 Friends and Followers

A key social concept in Twitter is the ability for one user to "follow" another user. The followed user's tweets will appear in the following user's home and friends timelines. To follow a user on behalf of the authenticating user, call the `FriendOperations`' `follow()` method:

```
twitter.friendOperations().follow("habuma");
```

Similarly, you may stop following a user using the `unfollow()` method:

```
twitter.friendOperations().unfollow("habuma");
```

If you want to see who a particular user is following, use the `getFriends()` method:

```
List<TwitterProfile> friends = twitter.friendOperations().getFriends("habuma");
```

On the other hand, you may be interested in seeing who is following a given user. In that case the `getFollowers()` method may be useful:

```
List<TwitterProfile> followers = twitter.friendOperations().getFollowers("habuma");
```

## 3.5 Twitter User Lists

In addition to following other users, Twitter provides the ability for users to collect users in lists, regardless of whether or not they are being followed. These lists may be private to the use who created them or may be public for others to read and subscribe to.

To create a new list, use `ListOperations`' `createList()` method:

```
UserList familyList = twitter.listOperations().createList(
       "My Family", "Tweets from my immediate family members", false);
```

`createList()` takes three parameters and returns a `UserList` object representing the newly created list. The first parameter is the name of the list. The second parameter is a brief description of the list. The final parameter is a boolean indicating whether or not the list is public. Here, false indicates that the list should be private.

Once the list is created, you may add members to the list by calling the `addToList()` method:

```
twitter.listOperations().addToList(familyList.getSlug(), "artnames");
```

The first parameter given to `addToList()` is the list slug (which is readily available from the `UserList` object). The second parameter is the screen name of a user to add to the list.

To remove a member from a list, pass the same parameters to `removedFromList()`:

```
twitter.listOperations().removeFromList(familyList.getSlug(), "artnames");
```

You can also subscribe to a list on behalf of the authenticating user. Subscribing to a list has the effect of including tweets from the list's members in the user's home timeline. The `subscribe()` method is used to subscribe to a list:

```
twitter.listOperations().subscribe("habuma", "music");
```

Here, `subscribe()` is given the list owner's screen name ("habuma") and the list slug ("music").

Similarly, you may unsubscribe from a list with the `unsubscribe()` method:

```
twitter.listOperations().unsubscribe("habuma", "music");
```

## 3.6 Searching Twitter

`SearchOperations` enables you to search the public timeline for tweets containing some text through its `search()` method.

For example, to search for tweets containing "#spring":

```
SearchResults results = twitter.searchOperations().search("#spring");
```

The `search()` method will return a `SearchResults` object that includes a list of 50 most recent matching tweets as well as some metadata concerning the result set. The metadata includes the maximum tweet ID in the search results list as well as the ID of a tweet that precedes the resulting tweets. The `sinceId` and `maxId` properties effectively define the boundaries of the result set. Additionally, there's a boolean `lastPage` property that, if `true`, indicates that this result set is the page of results.

To gain better control over the paging of results, you may choose to pass in the page and results per page to `search()`:

```
SearchResults results = twitter.searchOperations().search("#spring", 2, 10);
```

Here, we're asking for the 2nd page of results where the pages have 10 tweets per page.

Finally, if you'd like to confine the bounds of the search results to fit between two tweet IDs, you may pass in the since and maximum tweet ID values to `search()`:

```
SearchResults results = twitter.searchOperations().search("#spring", 2, 10, 145962,
 210112);
```

This ensures that the result set will not contain any tweets posted before the tweet whose ID is 146962 nor any tweets posted after the tweet whose ID is 210112.

## 3.7 Sending and receiving direct messages

In addition to posting tweets to the public timelines, Twitter also supports sending of private messages directly to a given user. `DirectMessageOperations`' `sendDirectMessage()` method can be used to send a direct message to another user:

```
twitter.directMessageOperations().sendDirectMessage("kdonald", "You going to the Dolphins
 game?")
```

`DirectMessageOperations` can also be used to read direct messages received by the authenticating user through its `getDirectMessagesReceived()` method:

```
List<DirectMessage> twitter.directMessageOperations().getDirectMessagesReceived();
```

`getDirectMessagesReceived()` will return the 20 most recently received direct messages.