Spring Social VKontakte Reference Manual

1.1.0.BUILD-SNAPSHOT

Craig Walls , Keith Donald

Table of Contents

1. Spring Social VKontakte Overview 1
    1.1. Introduction 1
    1.2. How to get 1
2. Configuring VKontakte Connectivity 2
3. VKontakte API Binding 4
    3.1. Retrieving a user's VKontakte profile data 4
    3.2. Getting a user's VKontakte connections 5

1. Spring Social VKontakte Overview

1.1 Introduction

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

VKontakte is a social networking site geared toward professionals. It enables its users to maintain and correspond with a network of contacts they have are professionally linked to.

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

1.2 How to get

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

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

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

Spring Social VKontakte uses Spring Social's `OAuth1Template` to add OAuth 1.0a authorization headers to requests sent to VKontakte. `OAuth1Template` uses the Commons Codec library when calculating request signatures. Therefore, you'll also need Commons Codec in your project:

```
<dependency>
  <groupId>commons-codec</groupId>
  <artifactId>commons-codec</artifactId>
  <version>1.5</version>
</dependency>
```

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

2. Configuring VKontakte 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 VKontakte provides `VKontakteConnectionFactory`, a `ConnectionFactory` for creating connections with VKontakte.

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

```
@Configuration
public class SocialConfig {

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

}
```

Here, a VKontakte 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 `VKontakteConnectionFactory`.

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 `VKontakteConnectionFactory` in XML:

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

  <bean class="org.springframework.social.vkontakte.connect.VKontakteConnectionFactory">
                <constructor-arg value="${vkontakte.consumerKey}" />
                <constructor-arg value="${vkontakte.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. VKontakte API Binding

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

To create an instance of `VKontakteTemplate`, you may pass in your application's OAuth 1 credentials, along with an access token/secret pair to the constructor:

```
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
VKontakte vkontakte = new VKontakteTemplate(consumerKey, consumerSecret, accessToken,
 accessTokenSecret);
```

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

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

    // ... use VKontakte API binding
}
```

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

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

3.1 Retrieving a user's VKontakte profile data

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

```
VKontakteProfile profile = vkontakte.getUserProfile();
```

The data returned in the `VKontakteProfile` includes the user's VKontakte ID, first and last names, their "headline", the industry they're in, and URLs for the public and standard profile pages.

If it's only the user's VKontakte ID you need, then you can get that by calling the `getProfileId()` method:

```
String profileId = vkontakte.getProfileId();
```

Or if you only need a URL for the user's public profile page, call `getProfileUrl()`:

```
String profileUrl = vkontakte.getProfileUrl();
```

3.2 Getting a user's VKontakte connections

To retrieve a list of VKontakte users to whom the user is connected, call the `getConnections()` method:

```
List<VKontakteProfile> connections = vkontakte.getConnections();
```

This will return a list of `VKontakteProfile` objects for the user's 1st-degree network (those VKontakte users to whom the user is directly linked--not their extended network).