This deliverable describes the plan for training 2nd open call experimenters in using the facility including explicit learning objectives, schedule and consortium responsibilities. The document provides all project partners with a description of what commitments will be needed to train users during the 2nd open call.
Project acronym         EXPERIMEDIA
Full title              Experiments in live social and networked media experiences
Grant agreement number  287966
Funding scheme          Large-scale Integrating Project (IP)
Work programme topic    Objective ICT-2011.1.6 Future Internet Research and Experimentation (FIRE)
Project start date      2011-10-01
Project duration        36 months

Activity 3              Operations
Workpackage 3.2         Experiment Support
Deliverable lead organisation  ICCS
Authors                 Athanasios Voulodimos (ICCS)  Konstantinos Psychas (ICCS)
Reviewers               Stephen C Phillips (IT Innov)
Version                 1.0
Status                  Final
Dissemination level     PU: Public
Due date                PM22 (2013-07-31)
Delivery date           2013-09-30

Note: there are two versions of this document. There is one which is confidential and contains the “EXPERIMEDIA Quick Start Guide” which has some private data in it. This version is public (as required by the contract) and does not contain the quick start guide.
# Table of Contents

1. Introduction .................................................................................................................................................. 4
2. Collection and evaluation of feedback from 1st open call experiments....................................................... 5
3. Updates in the training process ......................................................................................................................... 6
   3.1. EXPERIMEDIA Quick Start Guide ........................................................................................................ 6
   3.2. EXPERIMEDIA Troubleshooting Guide ................................................................................................. 6
   3.3. Training Workshop .................................................................................................................................. 7
4. Conclusion ...................................................................................................................................................... 8

Appendix A. Training Questionnaire ................................................................................................................ 9

Appendix B. Example Software Manual (SocialIntegrator) ............................................................................. 11
   B.1. Introduction ........................................................................................................................................... 11
   B.1.1. Dependencies between projects ........................................................................................................ 11
   B.1.2. Project releases ..................................................................................................................................... 12
   B.2. SocialIntegrator Components .................................................................................................................. 12
   B.2.1. SocialIntegrator Core ......................................................................................................................... 12
   B.2.2. SocialIntegrator Android .................................................................................................................. 20
   B.2.3. SocialIntegrator Web App for FHW ................................................................................................. 21
1. Introduction

This deliverable describes the plan for training second open calls experimenters on all aspects required to enable the latter to fully familiarise themselves with the particularities of the project, so as to appropriately design and conduct their experiments. The basis of the training strategy will remain similar to the one employed for the first open call experimenters; a detailed description of this was given in D3.2.2 “First EXPERIMEDIA training plan”. However, with the first round of open call experiments entering the final stages of their lifecycle, it’s only reasonable to collect and assess feedback from the partners who joined the consortium following the 1st open call and use that as input for redesigning the training process in a way that would best serve new partners.

It is important to stress that for the sake of brevity and simplicity this document will focus on merely describing updates to the first training plan, rather than repeating content. The remainder of this deliverable is structured as follows: Section 2 describes the process, results and conclusions drawn from collecting and evaluating feedback from the first open call experimenters regarding the training they underwent. In Section 3, we present the updates on the training process that appear as necessary; the feedback and advice of the first open call experimenters played a major role in defining the nature and details of these updates. Section 4 concludes the document and precedes the Appendices, which in this case constitute an integral part of the deliverable. The Appendices include (draft or example versions of) the additional material that will be supplied to the second open call experimenters to allow them to acquaint themselves with the project’s functioning details as fast as possible and to have easier access to the information they will be bound to need, whether it be of technical, high-level or logistic nature.
2. Collection and evaluation of feedback from 1st open call experiments

The first training plan targeted at the first open call experimenters joining EXPERIMEDIA at the beginning of Year 2 was created based on past experiences and on the particularities of the project. Although, judging from the results, the training process did attain its goals and the experimenters appeared to be generally satisfied with it, there are definitely parts that need or can be improved. However, this time we have an additional type of input, which is twofold: the experience of the core consortium who acted as the “trainers”, and, more importantly, the feedback from the first open call experimenters who underwent the training process. Being the most appropriate people to explain what the strong and weak aspects of the training (and support, as well), their feedback is deemed as very useful and will therefore constitute the most impactful parameter for the design of the second training plan.

Feedback was collected in three main ways:

- By evaluating comments and drawing related conclusions from the regular Activity 3 and 4 teleconferences,
- By discussing this issue face to face during a dedicated session of the General Assembly held in June 2013 and
- By creating a questionnaire to clarify some points that were not exhaustively covered through the above channels and to gain further insight.

One of the most significant points of criticism from the side of the experimenters was the fact that, even though most of the material required to understand the dynamics and particularities of the project existed, it was not sufficiently aggregated and, as a result, not always readily available. Furthermore, in some cases the material provided was more detailed and extensive than what would have been necessary to start with, at least. Therefore the creation of an “EXPERIMEDIA Quick Start Guide” (Section 3.1) arose as a salient update of the training plan.

Another update decided is the authoring of a “Troubleshooting Guide” to assist new experimenters in tackling “common” problems and/or point them to the appropriate person(s) in the project (Section 3.2). The feedback provided via the questionnaires also helped us redesign the training workshop so as to maximize its effectiveness (related details in Section 3.3).

The collection of feedback was accompanied by the provision of suggestions to the core consortium and “best practices” to the new experimenters expected to join the project after the 2nd open call. This acquired knowledge will be transferred to the new partners and will hopefully contribute to a faster and more profound assimilation in the project.

The template questionnaire that the first open call experimenters were asked to fill in is given in Appendix A.
3. Updates in the training process

3.1. EXPERIMEDIA Quick Start Guide

The creation of an “EXPERIMEDIA Quick Start Guide” is a crucial update of the training process in view of the second open call experimenters’ joining the project, as this will enable them to get up to speed fast and effectively.

The aim of the Quick Start Guide is to provide an initial unique point of reference to the new partners, concisely presenting all the information required for them to be integrated in the project both from a logistic and technical point of view, thus starting being productive as early on as possible.

The content that the Quick Start Guide should include has been discussed with the first open call experimenters, who have elaborated on their suggestions via the feedback questionnaires. Taking all of these into consideration, the Quick Start Guide should address the following topics (in descending order of priority, as this was expressed in the experimenters’ responses):

- Key persons in the project, a “who is who” list of names, roles and contact details of EXPERIMEDIA people
- Logistics of the project (meetings, teleconferences, deliverables’ processes, effort reporting, repositories, etc.)
- Technical details of the baseline technology
- High-level description of the functionality of the baseline components
- Project structure

Regarding the technical details, the information that is considered necessary includes:

- Specification of features and interfaces
- Current status of implementation, i.e., what has been implemented and what is in progress, along with a schedule of future releases and foreseen functionality of those
- Instructions on how to install, host and/or run the components
- Links to detailed documentation

An example of a software manual is given in Appendix B. The manual of the SocialIntegrator is a work in progress, and the version included in the Appendix does not address all the issues mentioned above.

A first draft of the EXPERIMEDIA Quick Start Guide is also provided in the confidential version of this document.

3.2. EXPERIMEDIA Troubleshooting Guide

Difficulties are bound to arise in any project, let alone a complex one such as EXPERIMEDIA. New partners unfamiliar with the particularities of the project may come at a point where they are not certain as to how they should proceed or whom they should turn to when they confront a problem. The EXPERIMEDIA Troubleshooting Guide will aim at either providing an answer
to questions that might arise in the course of the experimenters’ work or, otherwise, pointing them to the right direction for enquiry. Again, the valuable experience of the 1st open call experimenters has been employed to determine the occasions for which the assistance of the “core consortium” partners is necessary.

Based on this, a short list of such key occasions is the following:

- Difficulty with installing/using a piece of software
- Difficulty understanding the code of a baseline component; documentation is insufficient
- A partner is not responsive
- A deadline cannot be met

The EXPERIMEDIA Troubleshooting Guide will be ready for distribution among the 2nd open call partners at the Training Workshop (see Section 3.3 below).

### 3.3. Training Workshop

The agenda items for the Training Workshop of the 2nd open call experimenters have also been a topic of discussion among the consortium partners. Integral parts of this workshop, which is going to take place in the first weeks of the third year of the project, are considered:

- Brief presentations from the new experimenters, outlining their experiments and describing their initial questions
- Presentations of the functionality and use of the baseline components
- Technical demos of the components and the integrations between them
- Legal and ethical issues
- Venue related details
- Management issues

Apart from these, the possibility of special sessions will be given for discussions in several technical teams. Each technical team will be composed of a partner offering a technology (i.e. a Content Component owner) and the 2nd open call experimenters that intend to integrate this technology into their experiment. These sessions will provide the opportunity to experimenters and component owners for further comprehension (a) of the currently available functionality and (b) of the requirements for integrating the component in the experiments, respectively.

To maximize the benefit of the Workshop and of these sessions in particular, the technical teams will have been defined beforehand and a first contact among them will have been made, starting with the Kick-Off teleconferences.
4. Conclusion

The training of the new partners who join the consortium as a result of the Open Calls is of great significance in EXPERIMEDIA. At this second iteration of new partners joining the project, the consortium can make use of the lessons learnt from the experience of the first open call experiments. Suggestions, best practices and advice have been supplied by the first open call experimenters through a targeted questionnaire. This guidance was the main input in the process of updating the previous year’s training plan with a view to achieving a faster and more profound integration of the second open call partners into the consortium. The updates to this training plan have been described in this deliverable.
Appendix A. Training Questionnaire

EXPERIMEDIA

QUESTIONNAIRE ON OPEN CALL PARTNERS’ TRAINING PROCESS

This questionnaire is aimed at the partners of EXPERIMEDIA who joined the project as experimenters through the 1st Open Call. Its goal is to identify the changes that need to be made to the training plan, so as to improve the process in view of the 2nd Open Call. Your help is therefore valuable and greatly appreciated.

I. Quick Start Guide

During the Athens GA several OC experimenters shared the view that an “EXPERIMEDIA Quick Start Guide” would have been very helpful in introducing them to the project.

Below you will find some potential topics which could be included in such a Quick Start Guide. Please rate each topic on a scale of 1-5 (by underlining the appropriate number) with 1 being something you wouldn’t be interested in reading at the beginning stage and 5 being something you would definitely need to know about.

Short description (2 paragraphs) of what the project is about

Structure of the project (the five activities and who leads each)

Logistics of the project, e.g., what meetings are held and how often, telcos, who is supposed to participate, deliverable template, deliverable review process, quarterly effort reporting, repositories, Google Drive, etc.

Who is who – a list of names, roles and contact details of EXPERIMEDIA people

High-level description of the functionality of the components (similar to the content presented in the website)

Technical details

Please use the space below to write what specific technical details you would be interested in reading in such a guide (e.g. installation instructions, description of interfaces, etc.):  

Please write here any additional information that you consider essential and has not been mentioned above:

II. Troubleshooting Guide

We would like to identify what the appropriate content for an “EXPERIMEDIA OC Experimenters Troubleshooting Guide” would be.

Below you will find some potential problems, the remedy of which could belong to this guide. Please rate each issue on a scale of 1-5 (by underlining the appropriate number) with 1 being something that you do not regard as important and 5 being a high priority issue.
You are having difficulties with installing/using a piece of software. The issues are:
1. 2. 3. 4. 5.

You are having difficulties understanding the code of a baseline component; documentation is insufficient. The issues are:
1. 2. 3. 4. 5.

Another partner you are trying to contact is not responsive. The issues are:
1. 2. 3. 4. 5.

You have another issue but do not know whom you should contact. The issues are:
1. 2. 3. 4. 5.

Please write any additional problems (that you may or may not have faced during your experience in EXPERIMEDIA) which a troubleshooting guide should address:

III. Training workshop

A training workshop to help get new partners up to speed should cover a number of key aspects pertaining to the EXPERIMEDIA Facility. Integral parts would be presentation of the baseline technologies, technical demos of the components, venue-related details and legal and ethical issues.

Are there any topics not mentioned that you would consider necessary for such a workshop?

IV. Miscellaneous

Were you able to understand the interactions between the components? Would it have helped you if some components would be provided as a service? If yes, which components/applications would you have preferred to use a service?

Is there anything not covered in I, II & III above that you would consider an important part of training in EXPERIMEDIA?

Is there any “advice” or “best practice” that you would like to share with future experimenters who will join the consortium as a result of the 2nd open call?

Do you have any other remarks (positive or negative) related to your experience in EXPERIMEDIA both regarding training and in general, which you would like to share?

Thank you.
Appendix B. Example Software Manual (SocialIntegrator)

B.1. Introduction

SocialIntegrator software component extends SocialAuth\(^1\) and SocialAuth Android\(^2\) projects. The core of this project which is analysed in this document is a library that was used to develop an android application a web application and a monitoring client application. The last three projects were developed to support needs of the Foundation of the Hellenic World embedded experiment and their function will be discussed briefly. This component provides additional functionality for Facebook and Twitter but this document also provides insight for the use of SocialAuth in other social networks (SNs) as well.

B.1.1. Dependencies between projects

The diagram in Figure 1 presents the relations between different social components that were previously referenced. The relations are transitive and for presentation purposes not all lines were drawn. For example Social Android Application needs the libraries of all four projects (SocialAuth, SocialAuth Android, SocialIntegrator Core, SocialIntegrator Android) to work properly although some of them are used indirectly by the other libraries.

![Dependencies between social components](image-url)

**Figure 1** Dependencies between social components. They can be considered transitive

The dependencies just described imply that one can substitute the libraries of non-dependent project with newest ones as long as backward compatibility is kept. SocialAuth team has kept compatibility so far. Also NTUA plans to keep backwards compatibility as of version 1.3. It is

---

\(^1\) [https://code.google.com/p/socialauth/](https://code.google.com/p/socialauth/)

\(^2\) [https://code.google.com/p/socialauth-android/](https://code.google.com/p/socialauth-android/)
suggested to always use the latest release, as it contains bug fixes and new features, however SNs might still alter their API and make some features useless.

B.1.2. Project releases
In contrast with SocialAuth, that releases independent versions from their projects, SocialIntegrator releases simultaneously the versions of core and android projects which as a result have the same version number. Due to this fact we decided to include the classes of core library in android library so that one that wants to use the latter won’t need the first one.

The current releases of all projects on 16/06/2013 can be seen in Table 1.

<table>
<thead>
<tr>
<th>Project name</th>
<th>Latest release</th>
<th>Dependencies</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocialAuth</td>
<td>socialauth-java-sdk-4.2.zip</td>
<td>See dependencies folder of zip</td>
<td><a href="https://code.google.com/p/socialauth/downloads">https://code.google.com/p/socialauth/downloads</a></td>
</tr>
<tr>
<td>SocialIntegrator Core</td>
<td>SocialIntegrator-core-api-1.4alpha.zip</td>
<td>See lib folder</td>
<td><a href="https://digital.joanneum.at/svn/experimedia-sw/trunk/sn">https://digital.joanneum.at/svn/experimedia-sw/trunk/sn</a></td>
</tr>
<tr>
<td>SocialIntegrator Android</td>
<td>SocialIntegrator-android-api-1.4alpha.zip</td>
<td>See lib folder</td>
<td><a href="https://digital.joanneum.at/svn/experimedia-sw/trunk/sn">https://digital.joanneum.at/svn/experimedia-sw/trunk/sn</a></td>
</tr>
</tbody>
</table>

B.2. SocialIntegrator Components
In the following sections we are going to see each component and example uses of each one. Each component can be examined independently unless specified otherwise.

B.2.1. SocialIntegrator Core
The most important class of this component is SocialUtil\(^3\). It offers a way to add functionality to AuthProvider\(^4\) implementations of Facebook and Twitter that are the most popular without affecting them. Another way would be through plugins interfaces that SocialAuth developers recently added.

It can be seen that almost all exposed methods of this class has an AuthProvider parameter along with other parameters. This parameter should be an authorised Facebook or Twitter provider otherwise no result will be returned. Some methods still are not applicable to Twitter though. The next sections include important functions and examples of library.

\(^3\) gr.ntua.experimedia.socialintegrator.util.SocialUtil
\(^4\) org.brickred.socialauth.AuthProvider
B.2.1.1. Authorisation

The complete process of authorisation is described in the main page of SocialAuth project\(^5\). These steps are used in web application of FHW, described in B.2.3, for Facebook. For testing purposes alternatives can be used.

Facebook

Facebook provider uses OAuth2 protocol. Every request to the API should have a valid token. There are various kinds of tokens for Facebook\(^6\) but the most commonly used in applications are user tokens.

One way to obtain user tokens without registering an application is through signing in to open platforms like Facebook Explorer\(^7\) or Apigee\(^8\). The first one is official and gives you more flexibility for the token. It allows selecting custom permissions and it shows information about token with the debug tool. The latter requests access to all permissions to generate a token, which is only visible when you make a request. The token is contained in the url of any authenticated request as access_token=XXXXXXXX. This method on the other hand can be used for almost any SN.

It is evident that obtaining a token is not an automated process; however obtaining the app token does not require user interaction. Given the application's App ID and App secret from applications page\(^9\) one can use the getAppTokenFacebook method of SocialUtil like

```java
final String APP_ID = "XXX";
final String APP_SECRET = "YYY";
String token = SocialUtil.getAppTokenFacebook(APP_ID, APP_SECRET);
```

After that if you want to store the token you should do the following:

```java
SocialAuthConfig config = new SocialAuthConfig();
SocialAuthManager socialAuthManager = new SocialAuthManager();
InputStream is = new FileInputStream("oauth_configuration.properties");
//loads configuration of providers from file
//includes one or more providers but at least one should be Facebook
config.load(is);
socialAuthManager.setSocialAuthConfig(config);

AccessGrant agrant = new AccessGrant();
agrant.setKey(token);
agrant.setProviderId(Constants.FACEBOOK);
socialAuthManager.connect(agrant);
```

Then one can make requests to Facebook with this token automatically once he obtains the provider with:

```java
AuthProvider ap = socialAuthManager.getProvider(Constants.FACEBOOK);
```

\(^5\) [https://code.google.com/p/socialauth/#How_it_works](https://code.google.com/p/socialauth/#How_it_works)
\(^6\) [https://developers.facebook.com/docs/facebook-login/access-tokens/](https://developers.facebook.com/docs/facebook-login/access-tokens/)
\(^7\) [https://developers.facebook.com/tools/explorer/](https://developers.facebook.com/tools/explorer/)
\(^8\) [https://apigee.com/console/facebook](https://apigee.com/console/facebook)
\(^9\) [https://developers.facebook.com/apps](https://developers.facebook.com/apps)
**Twitter**

Twitter uses OAuth1 protocol. Twitter has a more detailed page\(^{10}\) on how to obtain tokens (of course it does not suggest Apigee\(^{11}\) console). For testing purposes the suggested option of accessing twitter from developer's account is the best\(^ {12}\). The code to store the access token will be a bit different though relatively to Facebook (the code to initialise `socialAuthManager` is the same) and is achieved with this method

```java
AccessGrant agrant = new AccessGrant();
String token = "XXX";
String secret = "YYY";
agrant.setKey(token);
agrant.setSecret(secret);
agrant.setProviderId(Constants.TWITTER);
socialAuthManager.connect(agrant);
```

The provider is then retrieved similarly with Facebook

```java
AuthProvider ap = socialAuthManager.getProvider(Constants.TWITTER);
```

**General Remarks**

Some points that apply to any provider (not only those mentioned above) are:

- Review the policy of the provider for access token, how soon they expire and what permissions they have. Don’t assume that you can use them indefinitely and for any request and always check the validity of the token if you don’t obtain it automatically.
- Register a dummy application to get its keys and put them in the configuration file. Even if you use tokens from another application you will still need the keys to make requests, which implies that the token doesn’t include information about the application that requested it.

**B.2.1.2. Requests to SNs’ API**

All SNs provide detailed documentation for their APIs and the limitations of using them. All these requests can be executed using a library implementing HTTP protocol. This process is automated with SocialAuth library and **almost** all requests can be executed using `api`\(^ {13}\) method offered by all provider implementations. SocialIntegrator automates most common requests but feel free to use this method to make a custom request if not one is available through the existing API. The signature of this method is:

```java
public Response api(String url, final String methodType,
        final Map<String, String> params,
        final Map<String, String> headerParams, final String body)
        throws Exception;
```

The names of the parameters are self-descriptive but the following comments give a bit more insight

---

\(^{10}\) [https://dev.twitter.com/docs/auth/obtaining-access-tokens](https://dev.twitter.com/docs/auth/obtaining-access-tokens)

\(^{11}\) [https://apigee.com/console/twitter](https://apigee.com/console/twitter)

\(^{12}\) [https://dev.twitter.com/docs/auth/tokens-devtwittercom](https://dev.twitter.com/docs/auth/tokens-devtwittercom)

\(^{13}\) org.brickred.socialauth.AuthProvider.api
• It should be mentioned that `parameters` variable is only used in POST and PUT requests. GET request parameters should be included in URL string like:
  
  ```
  http://www.example.com/api?param1=1&param2=a
  ```

• The documentation of each SN normally will instruct the user to use tokens as a parameter in the url in every request. `Api` function already does that so user does not need to worry about it. However, user may need to add a `&` manually at the end of any request involves parameters in url as `api` will not add it. Debugging information will reveal whether this issue is fixed.

• User should use null for parameters that are not needed which is usually the case for `params`, `headerParams` and `body`.

• The `Response` object contains information about the server's response like status code and response's body. The body of response can be obtained with `getResponseAsString` method of this class but it should be called when status code is 200(OK) otherwise it will likely throw Exception. The response format for most SNs is JSON but few of them (other than Facebook and Twitter) use XML.

• Parsing of response should be done with specialised libraries which are more robust, reliable and possibly more efficient\(^\text{14}\).

• User should include protocol in the url as it is not appended to it. Base url for Facebook graph API is "https://graph.facebook.com/" while for Twitter's new version of REST API is "https://api.twitter.com/1.1/". More on specific APIs of each SN follow.

A typical example that summarises some of the above points follows

```java
AuthProvider provider = socialAuthManager.getProvider(Constants.PROVIDER_NAME);
Response resp = provider.api("http://www.example.com/api?param1=1&param2=a&",
    MethodType.GET.toString(), null, null, null);
if (response.getStatus() == 200) {
    String respStr = resp.getResponseAsString(Constants.ENCODING);
    JSONObject respJSON = new JSONObject(respStr);
    ...}
```

**Facebook APIs**\(^\text{15}\)

Details for all Facebook APIs can be seen in the relative page in the official documentation. The most important API of all however is Graph API which can be used to query almost any information and most of the other APIs are based on that. FQL can come in handy if the programmer prefers SQL like queries and it is somewhat easier when he wants to get certain fields in the response. All other APIs have more limited use cases.

One of the simplest requests to graph API that returns the basic profile of the authenticated user(AU) is the following:

```java
Response resp = provider.api("https://graph.facebook.com/me",
    MethodType.GET.toString(), null, null, null);
```

\(^{14}\) [http://www.json.org/](http://www.json.org/)

\(^{15}\) [https://developers.facebook.com/docs/reference/api/](https://developers.facebook.com/docs/reference/api/)
**Twitter APIs**\(^{16}\)

Again Twitter developers' page provides all the information someone needs for the APIs. SocialIntegrator functions use the 1.1 version of the REST API and any implemented methods from SocialAuth have recently migrated to 1.1 as version 1 has been retired\(^{17}\).

Streaming APIs are useful only when an application needs constant access to a stream of tweets and they return data through an open HTTP request. These APIs return more information than REST but they are not supported yet and cannot be implemented with a single method.

One of the simplest requests to graph API that returns the latest tweets in AU’s timeline is:

```java
Response resp = provider.api("https://api.twitter.com/1.1/statuses/home_timeline.json", MethodType.GET.toString(), null, null, null);
```

### B.2.1.3. SocialIntegrator Requests

As already mentioned SocialIntegrator wraps the functionality of the most common SN requests in a common API. For this reason custom SN Objects are defined and returned by those requests. The objects defined follow. The text in parentheses refers to the section of appendix that has the documentation of the objects

- **AugmentedProfile**: this class extends original profile class of SocialAuth and basically it includes information about whether the user’s profile is a friend of the AU.
- **Event**: this class is only useful for Facebook as Twitter and most of the other SNs do not have the notion of the Event. However event might be implemented indirectly for Twitter. For example a hashtag may represent an event. The most important functions on event get the location and whether the AU attends the event. Date of the event will be added in the future.
- **Message**: it represents any piece of text published in a SN. ‘getMessage’ method returns these text while all other information might not be returned by every request and that can be related to the permissions the token has or simply is not returned.
- **Photo**: again not all methods might return something meaningful but the most important method is ‘getSource’ which returns a URL of the photo to be loaded. ‘getPicture’ returns a URL too, but it usually has lower analysis and is useful to be loaded as an icon in application.
- **Place**: this class represents a place object in a SN like a museum or restaurant and generally a point of interest. It includes another class Location which involves information about the place’s coordinates and address if applicable.

**Custom parsing**

All objects have a ‘getJSON’ method. In case a custom object of SocialIntegrator does not have a desired field or the API changes in a way that some fields are not parsed anymore, user can get the original server object with this method and parse in the way he likes. This is even useful if

---

\(^{16}\) [https://dev.twitter.com/docs](https://dev.twitter.com/docs)

\(^{17}\) [https://dev.twitter.com/calendar](https://dev.twitter.com/calendar)
the user wants to have the server response in the original format as there are already tools that work with JSON or XML objects.

The library used in this project is the same as SocialAuth’s and is included in dependencies. The library can be built from source or downloaded from sourceforge\textsuperscript{18}.

The JSON responses are not always included as they are not always needed and they take up memory. The methods of SocialUtil have usually a Boolean parameter that indicates whether the server response is included.

**B.2.1.4. Feeds and pagination**

Another issue that often arises when getting data from SNs is that sometimes they do not return all posts or all comments from a feed. The content of a feed cannot be easily determined due to synchronisation issues across servers and continuous updates in it. Even if it was reliable the amount of those data is prohibitive most of the times for a single request even for a great SN like Facebook. The solution is to get incrementally data once the user wants more. This is not an issue with a specific provider but it is the way to go with all of them. This is what SNs do on their own web site.

Both Facebook and Twitter introduce the notion of *pagination* on their developers’ pages\textsuperscript{19} with instructions on how to get data before and after the data returned from an initial request. The bulk of data returned resembles a page hence the name. The idea is to add extra parameters in request url that describe which is the most or least recent object previously returned so that the results returned are before or after that result.

SocialIntegrator makes it easier to work with timelines at least with Facebook methods. A new kind of generic list is introduced which includes two urls, one for the next page of results and one for the previous. It also has a method to retrieve the original list. This class is called SocialList.

**B.2.1.5. Photo upload**

The function of uploading photos is part of all providers basic APIs and is already implemented for Facebook and Twitter. This function is not necessarily supported by the API of other social networks so an Exception is thrown if it is not implemented or not supported.

This method however supports uploading image files only on AU’s profile. Facebook, on the other hand, supports image upload to other places as well, like events, pages and timelines of friends if friends allow it. To this direction SocialIntegrator uses a new provider class AugmentedFacebook to overcome this limitation. This class has an overloaded method upload images that has an additional argument for the id of the destination of the upload. As already mention this id can represent any facebook entity.

\textsuperscript{18} http://www.json.org/java/
http://sourceforge.net/projects/json-lib/files/

\textsuperscript{19} https://developers.facebook.com/docs/reference/api/pagination/
https://dev.twitter.com/docs/working-with-timelines
B.2.1.6. **SocialUtil examples**

Some common use case examples follow with inline comments.

**Initialise AugmentedFacebook provider**

The process is the same as in 0 except from the last line which should be:

```java
AugmentedFacebookImpl ap = (AugmentedFacebookImpl)
    socialAuthManager.getProvider(AugmentedFacebookImpl.CLASSID);
```

All the following examples require that `ap` is initialized properly either as Facebook or Twitter provider. Also the ids of pages or other objects are hardcoded so that they are easily accessible to the reader but it is suggested to load them from a properties file as it is easier to change them later.

**Get events from a Facebook page**

The page of this product was selected because it regularly updates and has active events. The status of the event is not available in this request but it would be if the source was “me”.

```java
String facebook_source = "cocacola";
//false indicates that json String will not be included in the objects of the list
SocialList<Event> events = SocialUtil.getEvents(ap, facebook_source, false);
//even though getEvents will not return null currently it is always safe to check
if (events != null) {
    //prints some information about each event
    for (Event event: events.getList()) {
        System.out.println("Event name:" + event.getName());
        System.out.println("Event location:" + event.getLocation());
    }
} else {
    System.out.println("No Events returned");
}
```

**Upload photo to a Facebook page**

This example requires that the provider object `ap` is an instantiation of `AugmentedFacebookImpl`. This example requires `publish_stream` permission to work. It would work the same if `facebook_source` was any page or event that allowed posting on its timeline. The uploaded image will have as description “This is an attempt to upload an image through my application”, while the third parameter of `uploadImage` is used only as a file identifier in debugging information.

```java
String facebook_source = "cocacola";
//the relative path of an image file
String imagepath = "/images/facebook_icon.png";
InputStream is = SampleFacebookApp.class.getResourceAsStream(imagepath);
try {
    Response resp = ap.uploadImage("This is an attempt to upload " +
        "an image through my application",
    facebook_source, "image", is);
    //Status shows whether the request succeeded
    //Notice that token should have publish stream permissions
    System.out.println("Status:" + resp.getStatus());
    System.out.println("Body:" +
        resp.getResponseBodyAsString(Constants.ENCODING));
```
Get likes of a photo and how many of those are from friends in Facebook

This example demonstrates several of the concepts discussed previously. First it gets all the friends of the AU so that the likes of friends are later recognized. Then it gets the likes of the Experimedia demo page for FHW giving the list of friends as argument. The request however returns only the first (25 current default) likes of the page. For all subsequent requests instead of getLikes method, the getLikesFromURL is used. This method uses the URL from the previous request that returns the next page of results. When there is no URL for the next page or no results are returned then there are no more likes. Lastly, due to the parameter of friends’ list the returned profiles can be queried about whether they are friends or not with the AU. The getLikes method is only implemented for Facebook as there is no straightforward way to get the respective of like on Twitter.

// Experimedia demo page for FHW
String facebook_source = "470432016341639";

// Gets authenticated user's friends
List<AugmentedProfile> l1 = SocialUtil.getFriends(ap, false);
for (AugmentedProfile augmentedProfile : l1) {
    System.out.println("Name:" + augmentedProfile.getFullName());
}

SocialList<AugmentedProfile> sl1 = SocialUtil.getLikes(ap, facebook_source, l1, true);

// Gets all likes following pagination links
// Warning! some objects might have millions likes
// and it's impossible to take them efficiently
List<AugmentedProfile> l2 = sl1.getList();
while ((sl1.getList().size()>0) && (sl1.nextPage!=null) && (sl1.nextPage!="")) {
    sl1 = SocialUtil.getLikesFromURL(ap, sl1.nextPage, l1, true);
    l2.addAll(sl1.getList());
}

// length of list is the number of likes
System.out.println("Likes:" + l2.size());

// for all likers print their name and whether they are your friends
for (AugmentedProfile augmentedProfile : l2) {
    System.out.println("Name:" + augmentedProfile.getFullName());
    System.out.println("Is friend? " +
            (augmentedProfile.isFriend() ? "Yes":"No"));
}

Get recent tweets that mention your name and have a photo in Twitter

This example shows the modularity of getMessages function and how it can be used in a complex Twitter task. Twitter has a special api endpoint that returns posts that mention the AU. Using this endpoint the current request returns the posts that contain a photo, which link is accessible with getLink function. Although it is rare a tweet might contain two or more pictures. In that case
the user has to make a custom parsing of the returned JSON of the tweet to get all of them by setting the last parameter of request to true and using getJSON function.

Also it appears that second parameter of getMessages is an empty string but it could be anything as it is taken into account only if the fourth parameter is CUSTOM_FEED. In this case the second parameter is the name of the feed which could be the id of a friend for Twitter or any object that has feed connection for Facebook.

```java
SocialList<Message> tweets = SocialUtil.getMessages(ap, "", ObjectType.PHOTO, FeedType.MENTION_FEED, false);
for (Message tweet : tweets.getList()) {
    System.out.println("Message:" + tweet.getMessage());
    System.out.println("Photo link:" + tweet.getLink());
}
```

**Make a search with specific parameters Twitter**

All parameters of the search query can be found in the original documentation\(^\text{20}\).

```java
String query = "potato";
Map<String,String> parameters = new HashMap<String,String>();
parameters.put("locale", "en");
parameters.put("count", "20");
parameters.put("since_id", "250075927172759552");
SocialList<Message> tweets = SocialUtil.search(ap, query, ObjectType.UNDEFINED, parameters, true);
for (Message tweet : tweets.getList()) {
    System.out.println("Message:" + tweet.getMessage());
    System.out.println("JSON String:" + tweet.getJson());
}
```

**B.2.2. SocialIntegrator Android**

This component facilitates the use of core functions of the previous component. The most important class of this component is SocialIntegratorAdapter\(^\text{21}\) which extends SocialAuthAdapter\(^\text{22}\) so the user can use methods of both.

**B.2.2.1. Authorisation**

The authorisation process is simpler than in the core project. The authorisation happens with a simple function implemented by SocialAuthAdapter, however one needs to pass a Listener class during initialization. The code to initialize this adapter is the following

```java
SocialIntegratorAdapter adapter = new SocialIntegratorAdapter(new ResponseListener());
```

where ResponseListener is

```java
private final class ResponseListener implements DialogListener {
    private static final String TAG = "ResponseListener";

    @Override
```

\(^{20}\) https://dev.twitter.com/docs/api/1.1/get/search/tweets

\(^{21}\) gr.ntua.experimedia.socialintegrator.android.extension.SocialIntegratorAdapter

\(^{22}\) org.brickred.socialauth.android.SocialAuthAdapter
public void onComplete(Bundle values) {
    /* Get the provider */
    final String providerName = values.getString(
            SocialIntegratorAdapter.PROVIDER);
    Log.d(TAG, "providername = " + providerName);

    /* Start new activity */
    Intent myIntent = new Intent(SocialIntegrator.this, NextView.class);
    startActivityForResult(myIntent, 0);
}

@Override
public void onError(SocialAuthError error) {
    Log.e(TAG, "Error: " + error.getMessage());
    error.printStackTrace();
}

@Override
public void onCancel() {
}

@Override
public void onBack() {
}
}

Then the following code will initiate a dialog that opens the provider's web page for
authorisation. This part authorises with Facebook and is usually added in a button listener while
its first parameter is the application's context

adapter.authorize(this, "facebook");

B.2.3. SocialIntegrator Web App for FHW

The web application used in FHW experiment uses the spring framework and the template of
SocialAuth23 for this framework. However, the applications of SocialAuth or SocialIntegrator are
not limited to this framework.

B.2.3.1. Functionality

The functionality of the application currently includes

1) Logging in with Facebook
2) See profile information of the AU
3) See and select events of a specific Facebook page (Experimedia demo page for FHW
currently24)
4) Join event if not already joined
5) See photos that are uploaded and questions & answers related with each one
6) Reply to any question

The application recognises messages that have a specific format. The functionality of the
application can be tested if there are images with a non empty description in the selected event.
Then questions are statuses that start with the description of the photo followed by ':' and then

23 https://code.google.com/p/socialauth/wiki/SpringSample
24 https://www.facebook.com/pages/Experimedia-Demo-Page-for-FHW/470432016341639
the question. These statuses are automatically generated in this format from the android application and recognised by this application that then removes the photo's description from each question.

The button that joins the event only appears when the user has not joined yet.

### B.2.3.2. Technical requirements

The application requires Java7 development kit\(^{25}\). The application was successfully deployed to VMWare server and Apache7. It is suggested to use one of those. In addition VMWare server is automatically configured with Spring Tool Suite download\(^{26}\).

### B.2.3.3. Application configuration

There is already information in a wiki of SocialAuth\(^{27}\) and in the README file of the committed application about configuring a new web application but here we are going to present the most important information applicable to any web application that uses Facebook.

Steps to register a new web application:

1) Go to Facebook developers page that manages applications\(^{28}\) and create a new application
2) Complete the website with facebook login option with the URL where the application will be actually deployed. Include the port number as well if it is not the default. The actual redirect URL is specified in root-context.xml of the application, must match with the one completed in this step and not end with the ’/’ character
3) In the advanced tab select the web option if it is not selected
4) Check regularly for changes in the API of Facebook and make sure the application resolves them.

The most important step is second one as described in SocialAuth's page\(^{29}\) the application has to redirect user to provider's (Facebook) page. The provider will then redirect back to the application's URL, after the user logins and grants permissions. Therefore this URL should be public and definitely not localhost. The template of this project takes care of parsing the token from redirect request and stores it for future use.


\(^{26}\) [http://www.springsource.org/sts](http://www.springsource.org/sts)

\(^{27}\) [https://code.google.com/p/socialauth/wiki/SpringSample](https://code.google.com/p/socialauth/wiki/SpringSample)

\(^{28}\) [https://developers.facebook.com/apps](https://developers.facebook.com/apps)

\(^{29}\) [https://code.google.com/p/socialauth/#How_it_works?](https://code.google.com/p/socialauth/#How_it_works?)