Deliverable 5.1 Identification of Social Network Providers and API Design
Final

Thomas Ludwig¹, Christian Reuter¹
University of Siegen¹
September 2014

Work Package 5

Project Coordinator
Prof. Dr.-Ing. Rainer Koch (University of Paderborn)

7th Framework Programme
for Research and Technological Development
COOPERATION
SEC-2013.6.1-1: The impact of social media in emergencies
<table>
<thead>
<tr>
<th>Distribution level</th>
<th>Public (PU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Due date</td>
<td>30/09/2014 (M6)</td>
</tr>
<tr>
<td>Sent to coordinator</td>
<td>30/09/2014</td>
</tr>
<tr>
<td>No. of document</td>
<td>D5.1</td>
</tr>
<tr>
<td>Title</td>
<td>Identification of Social Network Providers and API Design</td>
</tr>
<tr>
<td>Status &amp; Version</td>
<td>Final</td>
</tr>
<tr>
<td>Work Package</td>
<td>5: Information Collection and Presentation</td>
</tr>
<tr>
<td>Related Deliverables</td>
<td>D3.3, D4.1, D5.2, D5.4</td>
</tr>
<tr>
<td>Leading Partner</td>
<td>University of Siegen</td>
</tr>
</tbody>
</table>
| Leading Authors    | Thomas Ludwig, University of Siegen  
|                    | Christian Reuter, University of Siegen |
| Contributors       | Marc-André Kaufhold, University of Siegen  
|                    | Federico Sangiorgio, IES (section 5.3)  
|                    | Federica Toscano, IES (section 5.3)  
|                    | Massimo Cristaldi, IES (section 5.3)  
|                    | Mark Tolley, OCC (section 5.4 and 5.5)  
|                    | Mel Mason, OCC (section 5.4 and 5.5) |
| Reviewers          | Matthias Moi, University of Paderborn |
| Keywords           | Social Network Provider, API, Open Social, Activity Streams, Facebook, Twitter, Google+, YouTube, Tumblr, Instagram |

This project has received funding from the European Union’s Seventh Framework Programme for research, technological development and demonstration under grant agreement no 608352.
# Table of contents

List of Figures .................................................................................................................. V  
List of Tables ..................................................................................................................... VI  
Glossary .............................................................................................................................. VII  

1 **Introduction** ............................................................................................................. 1  
   1.1 Abstract .................................................................................................................. 1  
   1.2 Purpose of the document ...................................................................................... 1  
   1.3 Target audience ..................................................................................................... 1  

2 **Selection of Social Media Services** ....................................................................... 2  

3 **Existing Services for Data Gathering from Social Media** ..................................... 3  
   3.1 Requirements ........................................................................................................ 3  
   3.2 Existing Services .................................................................................................... 3  
      3.2.1 oneall ............................................................................................................... 3  
      3.2.2 socialmention ................................................................................................. 4  
      3.2.3 GNIP ................................................................................................................ 4  
      3.2.4 DataSift .......................................................................................................... 5  
      3.2.5 BlueJay .......................................................................................................... 5  
   3.3 Comparison ........................................................................................................... 5  

4 **Overall Social Media API Design** ....................................................................... 7  
   4.1 Crawl Service ........................................................................................................ 9  
   4.2 Extensibility regarding Platforms and Services ................................................... 9  

5 **Data Markup Language OpenSocial and Activity Streams** ................................. 11  
   5.1 OpenSocial Core and Social Data specification .................................................. 11  
   5.2 Activity Streams 2.0 specification ....................................................................... 12  
   5.3 OpenSocial Parser .............................................................................................. 13  

6 **Implementation of Social Media API** ............................................................... 15  
   6.1 Facebook and Twitter ........................................................................................... 15  
      6.1.1 Architecture and implementation .................................................................. 15  
      6.1.1.1 Data Storage: Object-relational mapping with Hibernate ....................... 15  
      6.1.1.2 Using the API-wrappers Facebook4J and Twitter4J ................................. 16  
      6.1.2 Package documentation of the Twitter/Facebook Access ............................ 16  
      6.1.3 Facebook and Twitter API endpoints ............................................................ 17
6.2 Twitter Firehose (not implemented) .................................................................................. 30

6.2.1 Comparison of Twitter Firehose and the Twitter Streaming API ................................. 30

6.2.2 The Firehose API ............................................................................................................. 31

6.2.3 Data Volumes .................................................................................................................. 32

6.3 Google Services .................................................................................................................. 32

6.3.1 YouTube Data API ......................................................................................................... 32

6.3.1.1 ActivityService ......................................................................................................... 33

6.3.1.2 ChannelService ...................................................................................................... 35

6.3.1.3 SearchService .......................................................................................................... 36

6.3.1.4 VideoService ............................................................................................................ 39

6.3.1.5 SubscriptionService ............................................................................................... 41

6.3.2 Google+ API .................................................................................................................. 43

6.3.2.1 PeopleService ........................................................................................................... 43

6.3.2.2 ActivityService ....................................................................................................... 45

6.3.2.3 CommentService .................................................................................................... 52

6.4 Tumblr (not implemented) .................................................................................................. 54

6.4.1 Tumblr REST API ......................................................................................................... 54

6.4.2 Tumblr Firehose ............................................................................................................ 55

6.4.3 Conclusions .................................................................................................................. 55

6.5 Instagram (not implemented yet) ....................................................................................... 55

6.5.1 Real-time API ................................................................................................................ 55

6.5.2 REST API ...................................................................................................................... 56

6.5.3 Conclusions .................................................................................................................. 56

7 Conclusion and Outlook ...................................................................................................... 57
List of Figures

Figure 1: OneAll (Source: http://docs.oneall.com/api/) ................................................................. 4
Figure 2: SocialMention (Source: http://www.socialmention.com) .................................................. 4
Figure 3: DataSift (Source: http://datasift.com/) ........................................................................ 5
Figure 4. Social Media API design as UML diagram ...................................................................... 7
List of Tables

Table 1. Evaluation of Requirement Fulfilment ................................................................................................. 6
Table 2. Social Media API services and paths .................................................................................................. 8
Table 3. Package documentation .................................................................................................................. 17
## Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Read, Update, Delete (Basic database operations)</td>
</tr>
<tr>
<td>EmerGent</td>
<td>Emergency Management in Social Media Generation</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>MAU</td>
<td>Monthly Active Users</td>
</tr>
<tr>
<td>OAuth</td>
<td>Open Standard for Authorization</td>
</tr>
<tr>
<td>OMA</td>
<td>Open Mobile Alliance</td>
</tr>
<tr>
<td>ORM</td>
<td>Object-Relational Mapping</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>UTF</td>
<td>Unicode Transformation Format</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Abstract

This deliverable presents the EmerGent Social Media API for continuously gathering data from various social media services. The document provides a documentation of the functionality and endpoints for accessing data from different social media services; this includes:

- a discussion about selected social media services the API focuses on
- a discussion of potentials of related technologies and APIs
- an overall API architecture design
- the data structure and its representation
- a technical functionality overview and documentation

1.2 Purpose of the document

The API aims at establishing a technical basis infrastructure for the following technical work packages. The detail targets are:

- Selection of a subsection of social media services this API focuses on
- Designing and implementing the technical API

In order to archive this purpose the following tasks have been performed:

- We have reviewed related approaches for gathering social media spanning data
- We have designed the overall architecture of the API
- We implemented and tested the API

1.3 Target audience

- Technical project consortium
- Software engineers
2 Selection of Social Media Services

In order to determine which social media services are to be used within this API implementation, the question has to be answered which platforms are currently the most relevant ones, although generally the system is designed platform-independently. Deliverable 3.3 [ReSc14] already outline those social network services with the highest MAU-Value (Monthly Active Users), which counts unique users over 30 days. In the context of EmerGent we focus on social media which are well-known in Europe. The first selection consists of Facebook (1.15 billion), YouTube (1 billion), WhatsApp (350 million), Google+ (327 million), Tumblr (300 million), Twitter (240 million), LinkedIn (184 million) and Instagram (150 million) [Ball13]. From this pre selection we further select a subset of these social network services where we tried to take at least one type of social media service into account (social network, microblogging, blogging, video- as well as photo-sharing social networks): After a second review we skipped LinkedIn, because it does not contain any crisis-relevant information. Furthermore we skipped WhatsApp because it does not provide any open API for implementing a data gathering services. Our subsection therefore consist of

- Facebook (social network)
- Google+ (social network)
- Twitter (microblogging)
- Tumblr (blogging)
- YouTube (video-based sharing)
- Instagram (photo-based sharing)

Twitter Firehose and Tumblr provide the data gathering just for a large amount of money. The Twitter Firehose is handled by two data providers¹, GNIP (owned by Twitter since April 2014, prices start at 500$/month) and DataSift (prices starting at 3000$/month²). Although we decided to design and implement the API as a first step based on free and open access social media APIs, we described the potential API access to the expensive Twitter Firehose and Tumblr also within this deliverable.

3 Existing Services for Data Gathering from Social Media

3.1 Requirements

As a first step we identify already existent approaches for the gathering of social media data that must fulﬁl the following requirements:

- **Multi-Platform Support**: Due to the high variety of social media services that are used during emergencies, we cannot specify in advance where the most relevant citizen-generated information will be. A requirement is therefore that a request allows access to multiple platforms.

- **Cross-Platform Usage**: The high variety of social media services requires a high variety of different accounts and information spaces. We do not exactly know which user uses which social media services. To reach most users a requirement is therefore to allow the posting and gathering of citizen-generated information spread widely on social media services.

- **Data Superiority (Own Data Storage and Access)**: To analyse citizen-generated information from social media services and treat them in an appropriated way (regarding privacy and ethical issues), a requirement is therefore that we need to store the data on our own servers. The own data storages provide us with access at any time without any limitations on requests or queries.

- **Crawl Service (Continuous gathering of social media data over a period of time)**: To continuously capture citizen-generated information during emergencies in nearly real-time, we need to specify a service that gathers the data over a deﬁned period of time. A requirement is therefore that crawl services are supported.

- **Location- and time-based data**: During emergencies location- and time-based information are very important, because they provide interesting context-data to the information itself. A requirement is therefore that location- and time-based data are provided with the information itself.

- **Not expensive**: An important requirement is that the access to social media services is not too expensive, as long as emergency services are not willing to pay huge amounts of data to access data.

3.2 Existing Services

3.2.1 oneall

oneall\(^1\) provides an API which unites more than 25 social media services and consolidates the most powerful social functions in one solution. The basic functionalities are free; for higher demands there are different comprehensive fee-required packages.

---

\(^1\) [http://www.oneall.com](http://www.oneall.com) [accessed: 2014/09/29]
After examining oneall we ascertain that only those social media posts can be accessed via the API, which have also been published with oneall. Since these posts represent only a very fractional part of the entire social media information space, we discard the option to make use of this provider.

3.2.2 socialmention

socialmention¹ is a search and analysis platform which aggregates and provides user-generated content from more than 100 different social networks based on a keyword search. An API enables the programmatic use of socialmention.

However, the search parameters required for our purposes are not sufficiently supported. First, the search interval can be specified only very vaguely and second, the system does not support any coordinate-based geo search.

3.2.3 GNIP

GNIP is a Colorado-based company that has recently been bought by Twitter. It provides a number of products including PowerTrack, a filtering API and Decahose, a “statistically valid” sample of at least 10% of all Tweets. GNIP provides access to a lot of social media services such as Tumblr, Foursquare, WordPress and Discus.

GNIP uses a subscription-based model with costs starting at $500 per month. In addition there is a charge of $0.1 per 1000 tweets.

3.2.4 DataSift

DataSift is a UK-based company that captures, augments and delivers social media data from a variety of sources including Twitter, Facebook, Google+, Instagram and Yammer. It has a single API based on a querying language. For Twitter, the output uses the same field names as the Twitter streaming API. The DataSift pricing model is complicated and is based in part on the amount of processing needed to implement a filter. Clients buy a number of Data Processing Units (DPU) either on demand or via a monthly subscription (costs opaque but appear to start at $2-3000 per month). There is a minimum charge of $0.20 per hour. In addition there is a charge of $0.1 per 1000 tweets.

![DataSift](http://datasift.com/)

*Figure 3: DataSift (Source: http://datasift.com/)*

3.2.5 BlueJay

BlueJay\(^1\) is a service allowing real-time monitoring of tweets for law enforcement services. It has access to the full Twitter Firehose (section 6.2) and a moderate subscription cost ($150/month). The interface appears basic and the company (Brightplanet.com) is American-owned.

Both DataSift and GNIP require approval from social media services for the exact use case and usually require a spend of at least $3000/month\(^2\) to make this worthwhile.

When a user deletes a tweet, a delete notice appears in the firehose stream. Clients using the firehose are required to delete the tweet on their systems.

3.3 Comparison

To sum up, there is a variety of cross-platform services to query social media, but all examined services lack in fulfilling the requirements for EmerGent. Table 1 depicts to what degree each service addresses the given requirements. While services like OneAll and SocialMentions have restrictions on data availability and filtering methods, the monthly fee of using GNIP or DataSift is way too expensive, if service availability has to be maintained over a long period of time. Implementing an own cross-platform service seems therefore the best solution for a) tailored the artefact for the EmerGent usage, b) maintain the functionality with internal

---

expertise and c) enabling the best adaptability and extensibility for changing or enhanced usages during EmerGent.

<table>
<thead>
<tr>
<th></th>
<th>OneAll</th>
<th>SocialMention</th>
<th>GNIP</th>
<th>DataSift</th>
<th>BlueJay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-Platform Support</td>
<td>good</td>
<td>Good</td>
<td>good</td>
<td>good</td>
<td>Twitter only</td>
</tr>
<tr>
<td>Cross-Plattform Usage</td>
<td>restricted</td>
<td>Restricted</td>
<td>good</td>
<td>good</td>
<td>Twitter only</td>
</tr>
<tr>
<td>Data superiority</td>
<td>Requires own implementation</td>
<td>Requires own implementation</td>
<td>US servers</td>
<td>US/UK servers</td>
<td>US servers</td>
</tr>
<tr>
<td>Crawl Service</td>
<td>not available</td>
<td>not available</td>
<td>good</td>
<td>good</td>
<td>restricted</td>
</tr>
<tr>
<td>Location and time-based data</td>
<td>restricted</td>
<td>restricted</td>
<td>good</td>
<td>Location query restricted</td>
<td>good</td>
</tr>
<tr>
<td>Not expensive</td>
<td>good</td>
<td>good</td>
<td>unacceptable</td>
<td>unacceptable</td>
<td>expensive</td>
</tr>
</tbody>
</table>

*Table 1. Evaluation of Requirement Fulfilment*
4 Overall Social Media API Design

Because current approaches are either too expensive or do not allow own data storages, we decided to design our own Social Media API as the base for following work packages and technical implementations.

The overall design of Social Media API follows multiple core elements and patterns, which are visualized in Figure 4 exemplarily for Facebook and Twitter and the service Message.

- **Platforms**: A platform is a social media service like Facebook, Google+ or Twitter functioning as a data provider for the Social Media API. Furthermore, the integration of a platform can be used to publish data to the regarding social media. Platforms are accessed or queried during the invocation of a service.

- **Services**: A service implements features of our Social Media API and processes requests of client applications or users. Each service provides a certain amount of methods that are oriented on CRUD\(^1\) operations.

---

\(^1\) CRUD operations are Create, Read (search), Update and Delete.
**Endpoints**: Each service must be accessible through an endpoint for client applications or users. In case of the Social Media API, endpoints are designed after the REST paradigm and are accessible through HTTP requests.

**Social entities**: A social entity is, for instance, a message, media item or person received from a social network. For processing and data storage, social data needs a standardized internal representation e.g. as objects.

To facilitate the integration and maintain extensibility of platforms and services, the implementation must follow certain patterns. First, each platform must implement a standardized platform interface. For that reason the platform factory can provide a consistent access to all platforms for a requesting service. Second, each service must provide an interface, so that each platform can implements its supported services. Hence, the service can access all platforms with the same methods. For each service, an endpoint will receive the HTTP requests and start the related service and service method. For reasons of clarity Figure 4 visualizes only a subset of possible platforms and services.

To add a new platform, it requires just the implementation of a) the platform interface and b) a mapping of the platform or its providing library to the relevant service interfaces. Table 2 visualizes services that are already available through the EmerGent Social Media API implementation (section 6.1) and their access through HTTP requests.

<table>
<thead>
<tr>
<th>Service</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crawl Service</td>
<td>GET crawlService/{requestId}/{startIndex}</td>
</tr>
<tr>
<td></td>
<td>POST crawlService</td>
</tr>
<tr>
<td></td>
<td>PUT crawlService</td>
</tr>
<tr>
<td></td>
<td>DELETE crawlService/{requestId}</td>
</tr>
<tr>
<td>Group Service</td>
<td>GET groupService/{groupId}</td>
</tr>
<tr>
<td></td>
<td>GET groupService/{groupId}/{requestId}/{startIndex}</td>
</tr>
<tr>
<td>Message Service</td>
<td>GET messageService/search/{keyword}</td>
</tr>
<tr>
<td></td>
<td>GET messageService/search/{requestId}/{startIndex}</td>
</tr>
<tr>
<td></td>
<td>GET messageService/{id}</td>
</tr>
<tr>
<td></td>
<td>POST messageService</td>
</tr>
<tr>
<td></td>
<td>DELETE messageService/{id}</td>
</tr>
<tr>
<td>Page Service</td>
<td>GET pageService/search/{pagId}</td>
</tr>
<tr>
<td>Person Service</td>
<td>GET peopleService/{id}</td>
</tr>
<tr>
<td></td>
<td>GET peopleService/{id}/messages</td>
</tr>
<tr>
<td></td>
<td>GET peopleService/{id}/{requestId}/{startIndex}</td>
</tr>
</tbody>
</table>

*Table 2. Social Media API services and paths*
Most endpoints specify further required or optional parameters that contain and filter the results. For example, a search for a social media message may be enhanced with parameters that contain the time, location or count of results (see section 6.1.3 for parameterized endpoint documentation). Towards this approach, the following is an exemplary service, which provides 50 messages from Facebook and Twitter that contain a specific keyword:

1. A client application or user starts a HTTP-GET request to the message endpoint: /messageService/getMessages?keyword=facebook,twitter&count=50
2. The request invokes the MessageEndpoint, which instantiates the correspondent MessageService.
3. The MessageService executes the getMessages (keyword, 50) function on the platforms Facebook and Twitter, if these platforms implement the MessageService.
4. The MessageService collects the results of both platforms and returns the combined result to the MessageEndpoint, but limited by the optional count parameter to a maximum of 50 results.
5. The MessageEndpoint returns the result as a JSON document to the requesting client application or user.

4.1 Crawl Service

A crawl service enables the client to capture contributions of current relevant events over a specific period of time. The client can instantiate multiple crawl service, as well as updating and deleting them (see section 6.1.3.11 for documentation). In a fixed interval the crawl service retrieves new posts from the source platforms and stores them into the database. Like this the API has the option of period-referenced retrievals additional to the retrievals referenced to a certain point of time.

4.2 Extensibility regarding Platforms and Services

To extend the Social Media API with further platforms, there are two steps to do for a successful integration.

a) Package platforms: Each platform must derive from platforms.AbstractPlatform which is placed in the same folder. At runtime, this package is searched for derivations, whereby the system recognizes which source platforms are supported. The class platforms.AbstractPlatform has abstract getters for the platforms’ name, its internal id and its service specified in opensocial.services. Those getters must be implemented for each derivation.

b) Package opensocial.services: For each service or endpoint, this package contains an interface with one or multiple method declarations. If a new platform needs to use one or more of the existing services, it has to implement the correspondent interfaces. The connection between service implementation and the new platform is realized through the platform-specific derivation of platforms.AbstractPlatform described above.
As a consequence, Java implementations of API services like Google+\(^1\), Instagram\(^2\), Tumblr\(^3\) or YouTube\(^4\) could be used to increase the scope of Social Media API. Because OpenSocial provides, for instance, a service specification for MediaItems\(^5\) like audio, images or videos, the package opensocial.services could be extended with interfaces that are required to implement a media service endpoint. That service could be used for platforms like Instagram or YouTube that have a stronger focus on media items.

For each platform a derivation of plattforms.AbstractPlattform in the same package has to be built (see section 0 for package documentation). At runtime this package is being searched for these derivations through reflection, whereby the system gets the information which source platforms are supported. The class plattforms.AbstractPlattform offers abstract getters for the platform’s name, the Id for internal use, and for every service specified in opensocial.services. They have to be implemented in the derivation.

The package opensocial.services contains an interface with one or more method headers for each service. If a platform is supposed to support a service, a platform-specific implementation of the corresponding interface has to be created. The connection between the service’s implementation and the platform is established through the platform-specific derivation of plattforms.AbstractPlattform as described in the previous section.

\(^1\) Google+ Java API, official library released under the ASL V2.0: [https://code.google.com/p/google-api-java-client/](https://code.google.com/p/google-api-java-client/) [accessed: 2014/09/29]

\(^2\) Instagram for Java, unofficial library released under the MIT License: [https://github.com/sola92/Instagram-Java](https://github.com/sola92/Instagram-Java) [accessed: 2014/09/29]

\(^3\) YouTube Data API Client Library for Java, official library released under the ASL V2.0: [https://developers.google.com/api-client-library/java/apis/youtube/v3](https://developers.google.com/api-client-library/java/apis/youtube/v3) [accessed: 2014/09/29]

\(^4\) Jumblr, official library released under the ASL V2.0: [https://github.com/tumblr/jumblr](https://github.com/tumblr/jumblr) [accessed: 2014/09/29]

\(^5\) OpenSocial service specification for MediaItems: [http://opensocial.github.io/spec/2.5.1/Social-API-Server.xml#MediaItems-Service](http://opensocial.github.io/spec/2.5.1/Social-API-Server.xml#MediaItems-Service) [accessed: 2014/09/29]
5 Data Markup Language OpenSocial and Activity Streams

In order to allow a general usage of our overall Social Media API we need to standardize our exchange data format. When we searched for existing cross-platform standards, especially concerning social media data structures, we decided to make use of the OpenSocial API formats due to its widely usage and deployments as well as its good documentation. The OpenSocial Foundation develops open standards in the context of social media and aims to break down technical barriers between different systems and to provide interoperability. In doing so, this organization – brought into life by Google – cooperates with its own community, the World Wide Web Consortium (W3C), the Open Mobile Alliance (OMA), and with various companies such as IBM and yahoo. Included among the standardized interfaces are those, which are relevant to the design of our API. First, OpenSocial specifies a REST-API\(^1\) for the consistent CRUD access to the resources of a social media platform and second, domain-specific core\(^2\) (array, boolean, collection, ...) and social media\(^3\) (group, message, person, Activity, ...) data structures are described in the form of entities and their relations to each other. Moreover, the activity data structure is based upon the Activity Streams 1.0 specification and therefore OpenSocial combines two different approaches, which will be discussed in the following sections.

5.1 OpenSocial Core and Social Data specification

The OpenSocial approach supports the definition of JSON and XML documents. Such documents may inherit a Collection as root entity, which again can be filled with a list of entities, such as Message or Person entities. Each entity has a defined set of attributes, for instance, a Person entity may contain a list of Address entities or a location in string representation. Therefore, to return a set of social media messages, the specification allows us to return a Collection of Message entity, whereof each contains a reference to the author represented as a Person entity. However, during the course of the implementation of a Social Media API we had to face several problems, which we will outline in the following.

Insufficient data structures: When transforming the posts and contributions made on social media services to the OpenSocial standard, it becomes apparent that the given data structures are not sufficient. For instance, the entity message lacks properties to describe likes or dislikes. In particular, there is no possibility to directly or indirectly tag a message with geo data. That is why a strict orientation towards this standard makes no sense for EmerGent. Instead, it must be enhanced by necessary structures.

Missing relations between entities: Missing relations between the entities is a further problem, which is closely related to the previous one. For example, the link between a

---

1 OpenSocial Social API Server Specification 2.5.1: [http://opensocial.github.io/spec/2.5.1/Social-API-Server.xml](http://opensocial.github.io/spec/2.5.1/Social-API-Server.xml) [accessed: 2014/09/29]

2 OpenSocial Core Data Specification 2.5.1: [http://opensocial.github.io/spec/2.5.1/Core-Data.xml](http://opensocial.github.io/spec/2.5.1/Core-Data.xml) [accessed: 2014/09/29]

3 OpenSocial Social Data Specification 2.5.1: [http://opensocial.github.io/spec/2.5.1/Social-Data.xml](http://opensocial.github.io/spec/2.5.1/Social-Data.xml) [accessed: 2014/09/29]
message and its author is not modelled as a direct relation; merely the Id of the author is saved in the message. Hence, the representation of relations through Ids must be enhanced by object references.

Id-entities: In order to identify resources, OpenSocial defines several id-entities. For instance, a person is identified with the aid of a User Id. Furthermore there are Group Id, App Id, Local Id and ObjectId. The reason for this differentiation lies in the specification of constants for frequent use cases. The User Id @me, for example, represents the current authenticated user, whereas @owner is the owner of a resource. The vast majority of these constants are not applicable for EmerGent because they focus on resource access within one platform which implements the standard. For the purpose of reducing complexity we go without this differentiation of id-entities and merely make use of ObjectId, which does not contain any pre-defined constants.

Inconsistent use of types: The high diversity of id-entities leads to their inconsistent use. That does not mean that the same entity is identified with varying id-entities in different contexts. What is meant here is the use of simple strings as identifier. For instance, the author of a message is identified with the string Sender Id, whereas users, in standard, are normally referenced with User Id. The reason for this inconsistent method does not become clear to us; that is why we substitute identifying strings by id-entities for the purpose of consistency.

To overcome these limitations and create an implementation that addresses our requirements, we had to extend the OpenSocial 2.5.1 specification. These alterations on the data structures imply that we do not comply with the specification, restricting the interoperability of this implementation. Therefore we further implemented the Activity Streams 2.0 support which a) seems more suitable to the usage withing EmerGent and b) does not change the related specification.

5.2 Activity Streams 2.0 specification

The OpenSocial 2.5.1 specification also integrates the Activity Streams 1.0 specification. Although not yet adapted into the specification, the OpenSocial team already provides Java and JavaScript implementations\(^1\) for creating, serializing and parsing Activity Streams 2.0 objects based on the JSON Activity Streams 2.0 draft 09\(^2\). The specification defines that “an activity is a semantic description of potential or completed actions”, which has at least a verb (the type of activity, e.g. like, post, share), an actor (e.g. the creator) and an object (e.g. an image or message object), but may also contain a target, participants and further attributes respectively objects. Each object may be enhanced with additional properties. There are already multiple verbs and object types defined within a specification\(^3\), for instance, a Place object may contain the attributes latitude, longitude and altitude. Although the specification

---


allows modelling the activities of liking, sharing and so on; there are no attributes designated to carry information like “20 users shared/liked this post”. While the specification may be extended with own verbs and object types, foreign implementations possibly have not enough knowledge to process them in the intended way. Therefore Activity Streams 2.0 overcomes the insufficiencies of the OpenSocial 2.5.1 data structures only to a certain degree. Activity objects must be encapsulated in a Collection object before serializing and returning them as a JSON object. Section 6.1.3.8 provides an exemplary output of an Activity Streams 2.0 JSON document through the Social Media API. As the OpenSocial 2.5.1 specification was implemented prior to Activity Streams 2.0, data is stored according the former specification and will be transformed into the latter, if desired.

The benefits of Activity Streams 2.0 are on the one hand, that each client built according the specification can automatically process the output of Activity Streams 2.0 data provider. On the other hand, the specification is developed alongside the Activity Streams 2.0 Action Handlers specification\(^1\), which provides standard mechanisms allowing activity streams applications to perform actions (e.g. like or share a video) without prior knowledge respectively custom code.

5.3 OpenSocial Parser

This section deals with how Java data classes are to be generated to match the OpenSocial standard. In order to avoid typing in the classes and their fields from the standard specification by hand and with the possibility in view of future updates of the standard, we decide to automate this process with the aid of a parser. The parser is realized as a Java application with graphical user interface. Beyond the automation this parser receives input from the responsible developer and supports him in overcoming the standard’s deficits described in section 4.1.

Analysis of the input document: The OpenSocial specification serves as input document which is available in the XML-format. Unfortunately the structure of the document does not reveal whether an element is a description of an entity respectively of a class or whether it is a content type irrelevant to the parser. That is why initially a class candidate is generated for every section of the input document.

Developer’s input: The results of the input document’s analysis are presented in tabular form. Afterwards the user can select those class candidates, from which Java classes are to be generated. For this, the application makes suggestions based on the results. To overcome the problem of different id-entities, there is an option to substitute all id-entities by one id-entity (Merge-id). Moreover, the user can display all fields of a class candidate. This display enables the resolution of the inconsistent use of simple strings as identifier to id-entities. Section 4.1 describes the lack of relations between objects in the OpenSocial standard. By clicking the “Create relations”\(^1\)-button the user can display all fields from a type of an id-entity of all class candidates. These fields are potential relations to other classes but can also serve as an identifier of the own class if it is a simple id-field. The user has the option to enhance these

fields by a direct relation to another object by modifying the corresponding row, especially the name and the type of the field. The latter must be the type of the referenced object.

**Output:** A Java class is generated for each class candidate, which has been selected by the user, in an own file in the package `opensocial.data`. Each class comprises appropriate fields, getters, setters, and comments. If a class has relations to other classes, the corresponding interfaces for the storage of these relations are implemented via hibernates (`HibernateDeepStorable` and `HibernateCheckForDuplicates`). Moreover, the creation of the annotations for hibernates and the JSON serialization is done automatically.

**Enhancement:** Manual enhancements of the generated classes by the programmer are done by derivations, so that the enhancements will not be overwritten as soon as the classes are regenerated. In doing so the missing fields and relations mentioned in section 4.1 can be enhanced without being affected by a later usage of the parser.
6 Implementation of Social Media API

6.1 Facebook and Twitter

6.1.1 Architecture and implementation

The current implementation of Facebook and Twitter access is written in Java and realized as Apache Tomcat server application (Figure 4). Using the Jersey reference implementation of Java API for RESTful Web Services, it provides REST-based endpoints to interact with any client. Endpoint results are returned as JSON documents, processed with the Jettison library. Internally, data structures according the OpenSocial 2.5.1 specification are realized with the OpenSocial Parser (section 0), and the Activity Streams 2.0 specification by the OpenSocial reference implementation for Java (section 5.2). The data is processed with the object-relational mapper Hibernate (section 6.1.1.1). While Twitter4J provides access to the Twitter REST API, Facebook4J enables the use of the Facebook Graph API in Java (section 6.1.1.2). Moreover, Social Media API provides interfaces for implementing other services such as Google+, Instagram or Tumblr.

Currently, the Social Media API provides five endpoints, whereof not each endpoint suits to each social network. While the CrawlService (section 4.1 for further information) and MessageService are intended to support all integrated networks, other endpoints are required to gather data specific to the correspondent social media, for instance, endpoints to query groups and pages in Facebook. This differentiation is required, because while groups and pages require an appropriate user token that grants access to the respective instance, the MessageService catches public feed messages without the requirement of a user token. Already processed requests can be reloaded without the need to query the related social media once again, which reduces a) the processing time of a request and b) the load of API query limits. Further details are specified within the endpoint documentation (section 6.1.3).

6.1.1.1 Data Storage: Object-relational mapping with Hibernate

For the purpose of storing and retrieving the collected data in and from a database, we deploy the Java framework Hibernate. With the aid of the ORM tools (object-relational mapper) we can go without direct database commands, because they are encapsulated in the framework. The generation of the database scheme as well the storage of the corresponding instances of the objects is done automatically. Only the annotations of the appropriate classes and their attributes are needed for Hibernate to transform the Java classes into database query

---

1 Apache Tomcat, released under the ASL V2.0: http://tomcat.apache.org/ [accessed: 2014/09/29]
2 Jersey, released under the CDDL V1.1 and GPL V2.0: https://jersey.java.net/ [accessed: 2014/09/29]
3 Jettison, released under the ASL V2.0: http://jettison.codehaus.org/ [accessed: 2014/09/29]
commands. In order to be able to consider future enhancements of the OpenSocial standard, the parser takes over the automatic generation of these annotations (section 0).

**Redundancy check:** The existing functionality of Hibernate has been enhanced by a redundancy check of certain objects in the database. At this, all classes are included which implement the Java interface `HibernateCheckForDuplicates`. This interface is implemented by the OpenSocial parser for all classes which contain an identifying `ObjectId`. The duplicate check is only done for objects of the same request.

**Problems with the database character set:** Despite the consequent use of UTF-8, the adding of some contributions lead to errors which indicate coding problems. After a time-consuming search for the problem we have come to the conclusion that the UTF-8-coding used by SQL does not fully support the entire character set, especially not the so-called emoticons. For a real UTF-8-coding the character set utf8mb4 has to be used. In doing so, several version-specific settings in the database and in the connection options of the JDBC driver have to be changed¹.

### 6.1.1.2 Using the API-wrappers Facebook4J and Twitter4J

After examining existing open source libraries for a Facebook or Twitter access, which makes it possible to address the APIs of the social media services, we select the API-wrappers facebook4j and twitter4j. The main reason for this selection is the fact that both are open source projects and currently being further developed. Thanks to the detailed documentation new functionalities can be implemented in a quite simple way, once the existing functionalities are not sufficient.

### 6.1.2 Package documentation of the Twitter/Facebook Access

<table>
<thead>
<tr>
<th>Package name</th>
<th>Package description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crawljob</td>
<td>Contains all classes and methods which are needed for crawljobs. The class <code>CrawlJobStarter</code> checks in intervals whether a resting crawljob is to be started.</td>
</tr>
<tr>
<td>facebook</td>
<td>Contains all classes and methods which are needed to implement the functions for facebook defined in opensocial.services package.</td>
</tr>
<tr>
<td>flexjson.custom</td>
<td>Contains all classes and methods which transform captured data into the JSON format (and vice versa).</td>
</tr>
<tr>
<td>hibernate</td>
<td>Contains all classes and methods which are needed to store and retrieve captured data in and from the database.</td>
</tr>
<tr>
<td>opensocial.</td>
<td>Contains all classes and methods which transform the OpenSocial</td>
</tr>
</tbody>
</table>

### 6.1.3 Facebook and Twitter API endpoints

#### 6.1.3.1 Structure of the IDs

Generally, the IDs from the source platforms are used for identifying resources of the platforms. An additional distinct domain, which corresponds to the source platform avoids the possible overlapping of IDs from different platforms.

**FORMAT**

```
{Domain}:{Id from source platform}
```

**EXAMPLES**

```
twitter:483960483939381248
facebook:100007226850470_1446575642259984
```

#### 6.1.3.2 Structure of the tokens

The source platforms demand an authentication through tokens for certain operations, for which there are different possibilities. The structure of the tokens is platform-specific. For the
transfer the token is a simple string, which, if necessary, is transformed to a structured object by the implementations of the respective platform. Which endpoints the transfer of the tokens requires, is shown in the documentation of the endpoints.

**FORMAT**

<table>
<thead>
<tr>
<th>Platform</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter</td>
<td>`{</td>
</tr>
<tr>
<td></td>
<td>&quot;oauth_token&quot;: &quot;.....&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;oauth_token_secret&quot;: &quot;.....&quot;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Facebook</td>
<td>&quot;.....&quot;</td>
</tr>
</tbody>
</table>

**6.1.3.3 Structure of the responses (Collection)**

The results of an API-request are delivered in a collection, which contains additional metadata. Requests, which merely deliver a requestId, are an exception, e.g. for asynchronous requests as well as GET platformService.

**RETURN PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>String</td>
<td>Id of the request, through which further requests can be made</td>
</tr>
<tr>
<td>status</td>
<td>String</td>
<td>Status of the request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Successful</strong> if the request has been successfully processed on the server,</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Executing</strong> if the process is still in progression, and <strong>Error if an error has</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>occurred.</strong></td>
</tr>
<tr>
<td>itemsPerPage</td>
<td>Integer</td>
<td>Number of the results which are maximally delivered for a request. This</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value is determined at the first request via the parameter <code>count</code>.</td>
</tr>
<tr>
<td>startIndex</td>
<td>Integer</td>
<td>Index of the first result</td>
</tr>
<tr>
<td>totalResults</td>
<td>Integer</td>
<td>Number of all available results. If the request has not yet been fully</td>
</tr>
<tr>
<td></td>
<td></td>
<td>processed on the server, this value becomes higher over the time.</td>
</tr>
<tr>
<td>list</td>
<td>Array</td>
<td>The actual list of results</td>
</tr>
</tbody>
</table>

**6.1.3.4 Synchronous and asynchronous endpoints**

If a request is sent to an API endpoint, the subsequent process will be carried out synchronously: The server will send an answer when all results internally processed. This mechanism could trigger a http-timeout and unnecessary latencies when processing big data sets, if, for instance, the client merely needs a subset of the results set, and only if required, receives further results. Therefore request-intensive endpoints allow an alternative asynchronous process, which is activated via the parameter `async`. Which endpoints support asynchronous processing can be viewed in the documentation of the endpoints. But by default and if not explicitly changed, requests are processed synchronous.

If there is an asynchronous request, the server will start its internal process in a new thread and will immediately send back the `requestId`. The results can then be retrieved though an appropriate endpoint by transferring this `requestId`. Here it should be noted that the results are provided for retrieval at an interval which has previously been determined by `count`. If, for
example, count is set to 20, the amount of the available results (*totalResults*) will be 0, 20, 40, 60... until all results are there.

### 6.1.3.5 Content-Type

If data are sent in the http-body of an http-request (in the following this is called payload), they are to be transformed into the JSON format. In addition, the corresponding content-type must be present in the http-header.

| Content-Type | application/json; charset=utf-8 |

### 6.1.3.6 PlattformService

The platform service provides information on the source platforms supported by the system.

- **GET plattformService**
  Provides information on the supported source platforms including the supported services

**URL, HTTP-METHOD**

plattformService via http-GET

**RETURN**

*Plattform[]*

**EXAMPLE FOR RETURN**

```json
[
  {
    "id": "twitter",
    "name": "Twitter",
    "supportedServices": [
      "MessageService",
      "PeopleService"
    ]
  },
  {
    "id": "facebook",
    "name": "Facebook",
    "supportedServices": [
      "MessageService",
      "GroupService",
      "PageService"
    ]
  }
]
```
6.1.3.7 GroupService

The GroupService enables the access to contributions from groups.

- GET groupService/{groupId}

Returns contributions from a group of a source platform

**URL, HTTP-METHOD**

`groupService/{groupId}` via http-GET

**PATH AND QUERY PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupId</td>
<td>String</td>
<td>required</td>
<td>Id of the group</td>
</tr>
<tr>
<td>token</td>
<td>String</td>
<td>required</td>
<td>Valid token of the platform</td>
</tr>
<tr>
<td>since</td>
<td>Long</td>
<td>optional</td>
<td>Minimum limit of the time period, in unix milliseconds</td>
</tr>
<tr>
<td>until</td>
<td>Long</td>
<td>optional</td>
<td>Maximum limit of the time period, in unix milliseconds</td>
</tr>
<tr>
<td>count</td>
<td>Integer</td>
<td>optional</td>
<td>Maximum number of results per request (default 50)</td>
</tr>
<tr>
<td>async</td>
<td>Boolean</td>
<td>optional</td>
<td>Indicates whether the request is processed synchronously or asynchronously (default false)</td>
</tr>
</tbody>
</table>

**RETURN**

`Collection<Message>` with startIndex 0

- GET groupService/{groupId}/{requestId}/{startIndex}

Serves the retrieval of results of `GET groupService/{groupId}` requests previously made (cf. ), which are identified by the `requestId`.

**URL, HTTP-METHOD**

`groupService/{groupId}/{requestId}/{startIndex}` via http-GET

**PATH PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupId</td>
<td>String</td>
<td>required</td>
<td>Id of the group</td>
</tr>
<tr>
<td>requestId</td>
<td>String</td>
<td>required</td>
<td>Id of the request previously made</td>
</tr>
<tr>
<td>startIndex</td>
<td>Integer</td>
<td>required</td>
<td>Index of the first result returned</td>
</tr>
</tbody>
</table>

**RETURN**

`Collection<Message>` with startIndex

6.1.3.8 MessageService

Via the MessageService contributions from the source platforms can be searched, retrieved, published and deleted. Messages will be returned either according the OpenSocial 2.5.1 or Activity Streams 2.0 specifications.
• GET `messageService/search/{keyword}`

Enables the keyword-based search for contributions. The determination of spatial and temporal limits for the results is possible by setting the optional parameters.

**URL, HTTP-METHOD**

`messageService/search/{keyword}` via http-GET

**PATH AND QUERY PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>keyword</em></td>
<td>String</td>
<td>required</td>
<td>Keyword for the search</td>
</tr>
<tr>
<td><em>platforms</em></td>
<td>String</td>
<td>required</td>
<td>Ids of the platforms to be searched. For more than one platform the Ids are separated by comma.</td>
</tr>
<tr>
<td><em>since</em></td>
<td>Long</td>
<td>optional</td>
<td>Minimum limit of the time period, in which the search takes place, in unix milliseconds</td>
</tr>
<tr>
<td><em>until</em></td>
<td>Long</td>
<td>optional</td>
<td>Maximum limit of the time period, in which the search takes place, in unix milliseconds</td>
</tr>
<tr>
<td><em>radius</em></td>
<td>Double</td>
<td>optional</td>
<td>Length of the search radius in km</td>
</tr>
<tr>
<td><em>latitude</em></td>
<td>Double</td>
<td>optional</td>
<td>Latitude for the geo search</td>
</tr>
<tr>
<td><em>longitude</em></td>
<td>Double</td>
<td>optional</td>
<td>Longitude for the geo search</td>
</tr>
<tr>
<td><em>count</em></td>
<td>Integer</td>
<td>optional</td>
<td>Maximum number of results per request (default 50)</td>
</tr>
<tr>
<td><em>async</em></td>
<td>Boolean</td>
<td>optional</td>
<td>Indicates whether the request is processed synchronously or asynchronously (default false)</td>
</tr>
<tr>
<td><em>activitystream</em></td>
<td>Boolean</td>
<td>optional</td>
<td>If true, returns activities according to the Activity Streams 2.0 specification</td>
</tr>
</tbody>
</table>

**RETURN**

Collection<Message> or Collection<Activity> with `startIndex 0`

**RETURN EXAMPLE FOR ACTIVITY STREAMS 2.0**

```json
{
    "totalItems": 2,
    "items": [
        {
            "verb": "post",
            "actor": {
                "objectType": "person",
                "id": "https://twitter.com/MissEllen93",
                "displayName": "MissEllen93",
                "content": "BTA :-) | großer Royaler Fan;) | Hobby Adelsexpertin | http://t.co/ACZCuDnLoF"
            },
            "object": {
                "objectType": "message",
```
GET messageService/search/{requestId}/{startIndex}

Serves the retrieval of results from search requests previously made (cf. ), which are identified by the requestId.

**URL, HTTP-METHOD**

messageService/search/{requestId}/{startIndex} via http-GET

**PATH PARAMETERS**
**RETURN**

*Collection*<Message*> with *startIndex*

- GET messageService/{id}

Enables the access to a single message, which is identified by the *id*

**URL, HTTP-METHOD**

messageService/{id} via http-GET

**PATH PARAMETER**

<table>
<thead>
<tr>
<th>id</th>
<th>String</th>
<th>required</th>
<th>Id of the message to be accessed</th>
</tr>
</thead>
</table>

**RETURN**

*Collection*<Message*> with exactly one item

- POST messageService

Publishes a message on a specified platform, whereat platform-specific valid tokens have to be transferred

**QUERY PARAMETER**

| platforms | String | required | Ids of the platforms, on which the message is to be published. For more than one platform the Ids are separated by comma. |

**PAYLOAD SCHEME**

```json
{
    "MESSAGE": {
        "BODY": "I AM A MESSAGE"
    },
    "TOKENS": [
        "TOKEN FOR THE FIRST PLATFORM",
        "TOKEN FOR THE SECOND PLATFORM"
    ]
}
```

The order of the tokens in the array has to correspond to the order of the platforms transferred as parameter! The first token is used for the first platform, the second token for the second platform and so on.

**RETURN**
The messages published on the platforms as *Collection<Message>* in the order, in which the platforms were transferred.

**PAYLOAD EXAMPLE**

```
{
    "MESSAGE": {
        "BODY": "HELLO! I AM A TEST MESSAGE"
    },
    "TOKENS": [
        "{"OAUTH_TOKEN":"XXX","OAUTH_TOKEN_SECRET":"YYY"},
        "ZZZ"
    ]
}
```

/SOCIALMEDIAAPI/MESSAGE SERVICE?PLATFORMS=TWITTER,FACEBOOK VIA HTTP-POST

**CORRESPONDING RETURN EXAMPLE**

```
{
    "itemsPerPage": 2,
    "list": [
        {
            "body": "Hello! I am a test message.",
            "id": "twitter:484234898518323202",
            "mediaItems": [],
            "num_comments": 0,
            "num_dislikes": 0,
            "num_likes": 0,
            "num_shares": 0,
            "num_views": 0,
            "num_votes": 0,
            "rating": 0,
            "recipientPersons": [],
            "recipients": [],
            "replies": [],
            "replyMessages": [],
            "sender": { },
            "senderId": "twitter:1053308640",
            "timeSent": 1404285571000
        },
        {
            "body": "Hello! I am a test message, too.",
            "id": "facebook:100001771091404_663335110402186",
            "mediaItems": []
        }
    ]
}
```
"num_comments": 0,
"num_dislikes": 0,
"num_likes": 0,
"num_shares": 0,
"num_views": 0,
"num_votes": 0,
"rating": 0,
"recipientPersons": [],
"recipients": [],
"replies": [],
"replyMessages": [],
"sender": {...},
}
"senderId": "facebook:100001771091404",
"status": "mobile_status_update",
"timeSent": 1404285571000,
"type": "status",
"updated": 1404285571000
}
]
"requestId": "4dec55970084a2ab4c607db6176f7ad48b1761da",
"startIndex": 0,
"status": "Successful",
"totalResults": 2
}

- DELETE messageService/{id}

Deletes a message from the source system. The tokens are transferred in the payload.

URL, HTTP-METHOD

messageService/{id} via http-DELETE

PATH PARAMETER

<table>
<thead>
<tr>
<th>id</th>
<th>String</th>
<th>required</th>
<th>Id of the message to be deleted</th>
</tr>
</thead>
</table>

PAYLOAD EXAMPLE

```
{
  "OAUTH_TOKEN": ".....",
  "OAUTH_TOKEN_SECRET": "....."
}
```

RETURN
A *Boolean*, which indicates whether the deletion was successful

### 6.1.3.9 PageService

Returns contributions from a source platform’s page

**URL, HTTP-METHOD**

`pageService/search/{pageld}` via http-GET

**PATH AND QUERY PARAMETERS**

<table>
<thead>
<tr>
<th>pageld</th>
<th>String</th>
<th>required</th>
<th>Id of the page</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td>String</td>
<td>required</td>
<td>Valid token of the platform</td>
</tr>
<tr>
<td>since</td>
<td>Long</td>
<td>optional</td>
<td>Minimum limit of the time period, in which the search takes place, in unix milliseconds</td>
</tr>
<tr>
<td>until</td>
<td>Long</td>
<td>optional</td>
<td>Maximum limit of the time period, in which the search takes place, in unix milliseconds</td>
</tr>
<tr>
<td>count</td>
<td>Integer</td>
<td>optional</td>
<td>Maximum number of results per request (default 50)</td>
</tr>
<tr>
<td>async</td>
<td>Boolean</td>
<td>optional</td>
<td>Indicates whether the request is processed synchronously or asynchronously (default false)</td>
</tr>
</tbody>
</table>

**RETURN**

`Collection<Message>` with `startIndex 0`

### 6.1.3.10 PeopleService

The PeopleService enables the access to user objects and to messages published by a user.

- **GET peopleService/{id}**

Returns the user including the `id` previously transferred

**URL, HTTP-METHOD**

`peopleService/{id}` via http-GET

**PATH PARAMETER**

<table>
<thead>
<tr>
<th>id</th>
<th>String</th>
<th>required</th>
<th>Id of the user to be accessed</th>
</tr>
</thead>
</table>

**RETURN**

`Collection<Person>` with exactly one item

- **GET peopleService/{id}/messages**

Enables the access to all messages of a user, where at a minimum and/or maximum limit can be optionally specified.

**URL, HTTP-METHOD**

`peopleService/{id}/messages` via http-GET
**PATH AND QUERY PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>required</td>
<td>Id of the user whose messages are to be accessed</td>
</tr>
<tr>
<td>sinceld</td>
<td>String</td>
<td>optional</td>
<td>Id of the first message to be returned</td>
</tr>
<tr>
<td>untilld</td>
<td>String</td>
<td>optional</td>
<td>Id of the last message to be returned</td>
</tr>
<tr>
<td>count</td>
<td>Integer</td>
<td>optional</td>
<td>Maximum number of results per request (default 50)</td>
</tr>
<tr>
<td>startIndex</td>
<td>Integer</td>
<td>optional</td>
<td>Index of the first result</td>
</tr>
<tr>
<td>async</td>
<td>Boolean</td>
<td>optional</td>
<td>Indicates whether the request is processed synchronously or asynchronously</td>
</tr>
</tbody>
</table>

**RETURN**

`Collection<Message>`

- GET `peopleService/{id}/{requestId}/{startIndex}`

Serves the retrieval of further messages of a user, after they have been loaded by GET `peopleService/{id}/messages` (cf. ). The path parameter `requestId` identifies the previous request.

**URL, HTTP-METHOD**

`peopleService/{id}/{requestId}/{startIndex}` via http-GET

**PATH PARAMETER**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>required</td>
<td>Id of the user whose messages are to be accessed</td>
</tr>
<tr>
<td>startIndex</td>
<td>Integer</td>
<td>optional</td>
<td>Index of the first result</td>
</tr>
<tr>
<td>requestId</td>
<td>String</td>
<td>required</td>
<td>Id of the request previously made</td>
</tr>
</tbody>
</table>

**RETURN**

`Collection<Message>`

**6.1.3.11 CrawlService**

The CrawlService enables archiving messages over a longer period of time, whereat all filter options are supported by GET `messageService/search/{keyword}`. If a running crawljob is interrupted by, for instance, the termination of the service or a crash, it will be continued after a reboot of the service.

- POST `crawlService`

  Initializes a new crawljob. If no start time is specified, the newly initialized crawljob will start immediately. If no end time is specified, the crawljob will run until it is terminated by DELETE `crawlService/{id}`.

**URL, HTTP-METHOD**

`crawlService` via http-POST
PAYLOAD PARAMETER

<table>
<thead>
<tr>
<th>parameter</th>
<th>type</th>
<th>mandatory</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword</td>
<td>String</td>
<td>required</td>
<td>Keyword, which is used for the search for messages to be archived</td>
</tr>
<tr>
<td>platforms</td>
<td>String[]</td>
<td>required</td>
<td>Ids of the platforms, in which the search takes place</td>
</tr>
<tr>
<td>Start</td>
<td>Long</td>
<td>optional</td>
<td>Start time of the archiving in unix milliseconds (default now)</td>
</tr>
<tr>
<td>End</td>
<td>Long</td>
<td>optional</td>
<td>End time of the archiving in unix milliseconds (default now)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If an end time is specified, it must be greater than the start time.</td>
</tr>
<tr>
<td>radius</td>
<td>Double</td>
<td>optional</td>
<td>Length of the search radius in km for the geo search for messages to be archived</td>
</tr>
<tr>
<td>latitude</td>
<td>Double</td>
<td>optional</td>
<td>Latitude for the geo search</td>
</tr>
<tr>
<td>longitude</td>
<td>Double</td>
<td>optional</td>
<td>Longitude for the geo search</td>
</tr>
<tr>
<td>count</td>
<td>Integer</td>
<td>optional</td>
<td>Maximum number of results per request (default 50)</td>
</tr>
<tr>
<td>waitBetweenRequests</td>
<td>Integer</td>
<td>optional</td>
<td>Time period between 2 search requests made by the crawljob in milliseconds (default 15<em>60</em>1000, at least 5<em>60</em>1000)</td>
</tr>
</tbody>
</table>

THE STRUCTURE OF THE PAYLOAD CAN BE VIEWED IN THE PAYLOAD EXAMPLE.

RETURN

The initialized crawljob¹

PAYLOAD EXAMPLE

```json
{
    "KEYWORD": "WORLDCUP",
    "PLATFORMS": [
        "FACEBOOK",
        "TWITTER"
    ],
    "COUNT": 75,
    "LATITUDE": 50.93383218959164,
    "LONGITUDE": 6.957178115844727,
    "RADIUS": 50,
    "WAITBETWEENREQUESTS": 1800000000,
    "END": 1404573448000
}
```

CORRESPONDING RETURN EXAMPLE

¹ see return example for structure
D5.1: Identification of Social Network Providers and API Design, Version V1, PU

```json
{
  "COUNT": 75,
  "CRAWLJOBID": "191116461203e0259d2124396a0374321b7e6fcb",
  "END": 1404573448000,
  "KEYWORD": "WORLDCUP",
  "LATITUDE": 50.9383218959164,
  "LONGITUDE": 6.957178115844727,
  "PLATFORMS": [
    "FACEBOOK",
    "TWITTER"
  ],
  "RADIUS": 50,
  "START": 1404228718059,
  "WAITBETWEENREQUESTS": 1800000
}
```

- **GET** `crawlService/{requestId}/{startIndex}`

Retrieves messages archived by the crawljob identified by `crawljobId`.

**URL, HTTP-METHOD**

`crawlService/{crawljobId}/{startIndex}` via http-GET

**PATH PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>crawljobId</code></td>
<td>String</td>
<td></td>
<td>Id of the crawljob</td>
</tr>
<tr>
<td><code>startIndex</code></td>
<td>Integer</td>
<td></td>
<td>Index of the first result</td>
</tr>
</tbody>
</table>

**RETURN**

`Collection<Message>` with `startIndex`.

- **PUT** `crawlService` with `startIndex`

Enables the subsequent editing of a crawljob, whereat merely the parameters presented below are supported. If a parameter is not transferred, it will be set to its standard value. The transfer of the identifying `crawljobId` is obligatory.

**URL, HTTP-METHOD**

`crawlService` via http-PUT

**PAYLOAD PARAMETERS**

29/58
THE STRUCTURE OF THE PAYLOAD IS ANALOGOUS TO THE PAYLOAD EXAMPLE ABOVE

RETURN

The edited `crawljob`

- DELETE `crawlService/{requestId}`

Terminates the `crawljob` identified by `requestId` and is to be used if the initialization of the `crawljob` has not set an end time.

URL, HTTP-METHOD

crawlService/{crawljobId} via http-DELETE

PFADPARAMETER

<table>
<thead>
<tr>
<th><code>crawljobId</code></th>
<th>String</th>
<th>required</th>
<th>Id of the <code>crawljob</code></th>
</tr>
</thead>
</table>

RETURN

The terminated `crawljob`¹

6.2 Twitter Firehose (not implemented)

This is a Twitter streaming API that guarantees near-real time delivery of all tweets matching the specified search criteria. It is not directly available from Twitter but can be accessed via partner providers DataSift² or Gnip³.

6.2.1 Comparison of Twitter Firehose and the Twitter Streaming API

The guaranteed delivery of all tweets corresponding to the search criteria is particularly important when monitoring and analysing tweets related to real-time events. The sampling system used by the streaming API appears to have biases that mean, for example, that the top

¹ structure analogous to the return example above
100 hashtags are misrepresented in the streaming API data\(^1\). It also noticeable that the streaming API sampling is reduced when the total tweet rate spikes (as would be expected during an emergency situation). There is a significant risk that valuable data about emergencies could be missed by using the streaming API, but on the one hand the other forms are too expensive, and on the other hand it is a research project and if all data available are required there is a need for a finance and business model.

6.2.2 The Firehose API

The firehose works in the same way as the other Twitter streaming APIs. A client makes a very long-lived HTTP request and parses the response incrementally. The connection is maintained as long as the client reads the data at a fast enough rate.

Messages are returned as JSON data structures such as the following:

```json
{
"created_at": "Mon Jun 27 13:31:06 +0000 2011",
"id": 85339370500014080,
"id_str": "85339370500014080",
"text": "test tweet",
"source": "<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>",
"truncated": false,
"in_reply_to_status_id": null,
"in_reply_to_status_id_str": null,
"in_reply_to_user_id": null,
"in_reply_to_user_id_str": null,
"in_reply_to_screen_name": null,
"user": {
"id": 324937307,
"id_str": "324937307",
"name": "MarkT",
"screen_name": "marktOCC",
"location": "",
"description": "",
"url": null,
"entities": {
"description": {
"urls": []
}
},
"protected": false,
"followers_count": 0,
"friends_count": 2,
"listed_count": 0,
"created_at": "Mon Jun 27 13:27:24 +0000 2011",
"favourites_count": 0,
"utc_offset": null,
"time_zone": null,
"geo_enabled": false,
"verified": false,
"statuses_count": 3,
"lang": "en",
"contributors_enabled": false,
"is_translation_enabled": false,
"profile_background_color": "C0DEED",
"profile_background_image_url": "http://abs.twimg.com/images/themes/theme1/bg.png",
```

---

\(^1\) [http://irevolution.net/2013/05/30/twitter-api-vs-firehose/](http://irevolution.net/2013/05/30/twitter-api-vs-firehose/) [accessed: 2014/09/29]
6.2.3 Data Volumes

These will obviously depend on the specificity of the filter terms, but a typical rate without filters would be 7,500 tweets per second\(^1\).

6.3 Google Services

6.3.1 YouTube Data API

The YouTube Data API lets you incorporate functions normally executed on the YouTube website into your own website or application. The API interacts with the following types of resources:

**ACTIVITIES**

An activity resource contains information about an action that a particular channel, or user, has taken on YouTube. The actions reported in activity feeds include rating a video, sharing a video, marking a video as a favorite, commenting on a video, uploading a video, and so forth. Each activity resource identifies the type of action, the channel associated with the action, and the resource(s) associated with the action, such as the video that was rated or uploaded.

---

A channel resource contains information about a YouTube channel.

**SEARCH**

A search result contains information about a YouTube video, channel, or playlist that matches the search parameters specified in an API request. While a search result points to a uniquely identifiable resource, like a video, it does not have its own persistent data.

**VIDEOS**

A video resource represents a YouTube video.

**SUBSCRIPTIONS**

A subscription resource contains information about a YouTube user subscription. A subscription notifies a user when new videos are added to a channel or when another user takes one of several actions on YouTube, such as uploading a video, rating a video, or commenting on a video.

6.3.1.1 ActivityService

The ActivityService returns a list of channel activity events that match the request criteria. It enables retrieving events associated with a particular channel, events associated with the user's subscriptions and Google+ friends, or the YouTube home page feed, which is customized for each user.

- GET /youtube/ActivityService/{channelId}

Retrieves events associated with a particular channel. Every request must specify an API key (with the key parameter).

**PATH PARAMETERS**

| channelId | string | required | specifies a unique YouTube channel ID. |

**QUERY PARAMETERS**

<table>
<thead>
<tr>
<th>part</th>
<th>string</th>
<th>optional</th>
<th>specifies a comma-separated list of one or more activity resource properties that the API response will include. (default snippet)</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>required</td>
<td>specifies an API key</td>
</tr>
<tr>
<td>maxResults</td>
<td>unsigned integer</td>
<td>optional</td>
<td>specifies the maximum number of items that should be returned in the result set. Acceptable values are 0 to 50, inclusive. (default 5)</td>
</tr>
<tr>
<td>pageToken</td>
<td>string</td>
<td>optional</td>
<td>identifies a specific page in the result set that should be returned. In an API response, the nextPageToken and prevPageToken properties identify other pages that could be retrieved.</td>
</tr>
</tbody>
</table>
### RESPONSE EXAMPLE

This method returns a response with a structure similar to the following example:

```json
{
  "etag": "\"kjEFmP90GvrCl88ObMQGorfgaQ/9bhMRjiZz-U068HWOabRj2bglq8\"",
  "id": "VTE0MTA0MDk1NzcxMzk2OTMzMDM0NDExNjg=",
  "kind": "youtube#activity",
  "snippet": {
    "channelId": "UCE_M8A5yxnLfwOgEeajjw",
    "channelTitle": "Apple",
    "description": "From the launch of Apple Watch to the arrival of iPhone 6 to a live performance from U2, this is an event not to be missed.\n\nhttp://www.apple.com/live/2014-sept-event/",
    "publishedAt": "2014-09-11T04:26:17.000Z",
    "thumbnails": {
      "default": {
        "height": 90,
        "url": "https://i.ytimg.com/vi/38IqQpwPe7s/default.jpg",
        "width": 120
      },
      "high": {
        "height": 360,
        "url": "https://i.ytimg.com/vi/38IqQpwPe7s/hqdefault.jpg",
        "width": 480
      },
      "maxres": {
        "height": 720,
        "url": "https://i.ytimg.com/vi/38IqQpwPe7s/maxresdefault.jpg",
        "width": 1280
      }
    }
  }
}
```
6.3.1.2 ChannelService

The ChannelService returns a collection of zero or more channel resources that match the request criteria.

- GET /youtube/ChannelService/{id}

This request retrieves information about the YouTube channel identified by the {id} parameter. Every request must specify an API key (with the key parameter).

**PATH PARAMETERS**

| id | string | required | specifies a comma-separated list of the YouTube channel ID(s) for the resource(s) that are being retrieved. |

**QUERY PARAMETERS**

<table>
<thead>
<tr>
<th>part</th>
<th>string</th>
<th>optional</th>
<th>specifies a comma-separated list of one or more channel resource properties that the API response will include. (default snippet)</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>required</td>
<td>specifies an API key</td>
</tr>
<tr>
<td>maxResults</td>
<td>unsigned integer</td>
<td>optional</td>
<td>specifies the maximum number of items that should be returned in the result set. Acceptable values are 0 to 50, inclusive. (default 5)</td>
</tr>
</tbody>
</table>
RESPONSE EXAMPLE

This method returns a response with a structure similar to the following example:

```json
{
  "etag": "\"kjEFmP90GvrCl8BObMQtGoRfqaQ/gyZlNERvs0wUzPa2AW_QWqGQA4c\"",
  "id": "UCE_M8A5yxnLf0KghEeajjw",
  "kind": "youtube#channel",
  "snippet": {
    "description": "Apple designs the Mac, along with OS X, iLife, and iWork. It leads the digital music revolution with iPods and iTunes. It reinvented the mobile phone with iPhone and App Store. And it's defining the future of mobile media and computing with iPad."
  },
  "publishedAt": "2005-06-22T05:23:00Z",
  "thumbnails": {
    "default": {
      "url": "https://yt3.ggpht.com/-KdgJnz1HldQ/AAAAAAAI/AAAAAAAI/4vVN7sljq4/s88-c-k-no/photo.jpg"
    },
    "high": {
      "url": "https://yt3.ggpht.com/-KdgJnz1HldQ/AAAAAAAI/AAAAAAAI/4vVN7sljq4/s240-c-k-no/photo.jpg"
    },
    "medium": {
      "url": "https://yt3.ggpht.com/-KdgJnz1HldQ/AAAAAAAI/AAAAAAAI/4vVN7sljq4/s240-c-k-no/photo.jpg"
    }
  },
  "title": "Apple"
}
```

6.3.1.3 SearchService

The SearchService returns a list of search results that contains information about a YouTube video, channel, or playlist that matches the search parameters specified in the API request.

- **GET /youtube/SearchService/(order)/(location)/(locationRadius)**

This request retrieves a collection of search results included in a circular geographic area identified by the location and locationRadius parameters. The order parameter specifies the...
method to use to sort results. The default value is relevance. Every request must specify an API key (with the key parameter).

**PATH PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>order</td>
<td>string</td>
<td>required</td>
<td>specifies the method that will be used to order resources in the API request. (default relevance)</td>
</tr>
<tr>
<td>location</td>
<td>string</td>
<td>required</td>
<td>the location parameter, in conjunction with the locationRadius parameter, defines a circular geographic area and also restricts a search to videos that specify, in their metadata, a geographic location that falls within that area.</td>
</tr>
<tr>
<td>locationRadius</td>
<td>string</td>
<td>required</td>
<td>The locationRadius parameter, in conjunction with the location parameter, defines a circular geographic area.</td>
</tr>
</tbody>
</table>

**QUERY PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>part</td>
<td>string</td>
<td>optional</td>
<td>specifies a comma-separated list of one or more search resource properties that the API response will include. (default snippet)</td>
</tr>
<tr>
<td>key</td>
<td>string</td>
<td>required</td>
<td>specifies an API key</td>
</tr>
<tr>
<td>maxResults</td>
<td>unsigned integer</td>
<td>optional</td>
<td>specifies the maximum number of items that should be returned in the result set. Acceptable values are 0 to 50, inclusive. (default 5)</td>
</tr>
<tr>
<td>q</td>
<td>string</td>
<td>optional</td>
<td>specifies the query term to search for.</td>
</tr>
<tr>
<td>pageToken</td>
<td>string</td>
<td>optional</td>
<td>identifies a specific page in the result set that should be returned. In an API response, the nextPageToken and prevPageToken properties identify other pages that could be retrieved.</td>
</tr>
<tr>
<td>publishedAfter</td>
<td>datetime</td>
<td>optional</td>
<td>indicates that the API response should only contain resources created after the specified time. The value is an RFC 3339 formatted date-time value (1970-01-01T00:00:00Z).</td>
</tr>
<tr>
<td>publishedBefore</td>
<td>datetime</td>
<td>optional</td>
<td>indicates that the API response should only contain resources created before the specified time. The value is an RFC 3339 formatted date-time value (1970-01-01T00:00:00Z).</td>
</tr>
</tbody>
</table>
GET /youtube/SearchService/{order}

This request retrieves a list of search results sorted in the order specified by the order parameter. The default value is relevance. Every request must specify an API key (with the key parameter).

**PATH PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>order</td>
<td>string</td>
<td>required</td>
<td>specifies the method that will be used to order resources in the API request. (default relevance)</td>
</tr>
</tbody>
</table>

**QUERY PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>part</td>
<td>string</td>
<td>optional</td>
<td>specifies a comma-separated list of one or more search resource properties that the API response will include. (default snippet)</td>
</tr>
<tr>
<td>key</td>
<td>string</td>
<td>required</td>
<td>specifies an API key</td>
</tr>
<tr>
<td>maxResults</td>
<td>signed integer</td>
<td>optional</td>
<td>specifies the maximum number of items that should be returned in the result set. Acceptable values are 0 to 50, inclusive. (default 5)</td>
</tr>
<tr>
<td>q</td>
<td>string</td>
<td>optional</td>
<td>specifies the query term to search for.</td>
</tr>
<tr>
<td>pageToken</td>
<td>string</td>
<td>optional</td>
<td>identifies a specific page in the result set that should be returned. In an API response, the nextPageToken and prevPageToken properties identify other pages that could be retrieved.</td>
</tr>
<tr>
<td>publishedAfter</td>
<td>datetime</td>
<td>optional</td>
<td>indicates that the API response should only contain resources created after the specified time. The value is an RFC 3339 formatted date-time value (1970-01-01T00:00:00Z).</td>
</tr>
<tr>
<td>publishedBefore</td>
<td>datetime</td>
<td>optional</td>
<td>indicates that the API response should only contain resources created before the specified time. The value is an RFC 3339 formatted date-time value (1970-01-01T00:00:00Z).</td>
</tr>
</tbody>
</table>

**RESPONSE EXAMPLE**

These requests return a response with a structure similar to the following example:

```json
{
  "etag": "\"kjEFmP90GvrCl8BObMQtGofgaQ/3E5CajPp9uYQHfGok1CdJzuBeA4\"",
  "id": {
    "kind": "youtube#video",
```
6.3.1.4 VideoService

The VideoService returns a list of videos that match the API request parameters.

- **GET /youtube/VideoService/{id}**

This request retrieves the list of videos that match the YouTube video ID(s) specified in the {id} parameter. Every request must specify an API key (with the key parameter).

**PATH PARAMETERS**

| id      | string | required | specifies a comma-separated list of the YouTube video ID(s) for the resource(s) that are being retrieved. |

**QUERY PARAMETERS**
D5.1: Identification of Social Network Providers and API Design, Version V1, PU

<table>
<thead>
<tr>
<th>part</th>
<th>string</th>
<th>optional</th>
<th>specifies a comma-separated list of one or more video resource properties that the API response will include. (default snippet)</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>required</td>
<td>specifies an API key</td>
</tr>
<tr>
<td>maxResults</td>
<td>unsigned integer</td>
<td>optional</td>
<td>specifies the maximum number of items that should be returned in the result set. Acceptable values are 1 to 50, inclusive. (default 5)</td>
</tr>
<tr>
<td>pageToken</td>
<td>string</td>
<td>optional</td>
<td>identifies a specific page in the result set that should be returned.</td>
</tr>
</tbody>
</table>

**RESPONSE EXAMPLE**

This method returns a response with a structure similar to the following example:

```json
{
    "etag": "\"kjEFmP90GvrCl8BObMQtGoRfqaQ/1s-soKubfQl9jg2EMfEDwIrwF0\"",
    "id": "FglqN1jd1tM",
    "kind": "youtube#video",
    "snippet": {
        "categoryId": "28",
        "channelId": "UCE_M8ASyxnLfw0KghEeajw",
        "channelTitle": "Apple",
        "description": "Featuring a larger, more advanced display, and significant leaps in capability and performance, iPhone 6 and 6 Plus represent the biggest advancement in design and engineering since we introduced the original iPhone.\n\nhttp://www.apple.com/iphone-6/?cid=www-us-yt-ip6-pm",
        "liveBroadcastContent": "none",
        "publishedAt": "2014-09-09T20:23:07.000Z",
        "thumbnails": {
            "default": {
                "height": 90,
                "url": "https://i.ytimg.com/vi/FglqN1jd1tM/default.jpg",
                "width": 120
            },
            "high": {
                "height": 360,
                "url": "https://i.ytimg.com/vi/FglqN1jd1tM/hqdefault.jpg",
                "width": 480
            }
        },
        "maxres": {
```
6.3.1.5 SubscriptionService

The SubscriptionService returns subscription resources that match the API request criteria.

- **GET /youtube/SubscriptionService/{channelId}**

  The request returns the list of subscriptions to the channel that match the YouTube channel ID specified in the {id} parameter. Every request must specify an API key (with the key parameter).

**PATH PARAMETERS**

<table>
<thead>
<tr>
<th>channelId</th>
<th>string</th>
<th>required</th>
<th>specifies a YouTube channel ID. The API will only return that channel's subscriptions.</th>
</tr>
</thead>
</table>

**QUERY PARAMETERS**

<table>
<thead>
<tr>
<th>part</th>
<th>string</th>
<th>optional</th>
<th>specifies a comma-separated list of one or more subscription resource properties that the API response will include. (default snippet)</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>required</td>
<td>specifies an API key</td>
</tr>
<tr>
<td>maxResults</td>
<td>unsigned integer</td>
<td>optional</td>
<td>specifies the maximum number of items that should be returned in the result set. Acceptable values are 0 to 50, inclusive. (default 5)</td>
</tr>
</tbody>
</table>
RESPONSE EXAMPLE

This method returns a response with a structure similar to the following example:

```json
{
  "etag": "\"kjEFmP90GvC18B0bMQrtGoRfgaQ/5rFvd051HP_X1Tex8FlnQTP3QM\"",
  "id": "Y3Vhs7MADFj1vDwd1P1mi3_J4m61VMCKJ8FDD9ryGc",
  "kind": "youtube#subscription",
  "snippet": {
    "channelId": "UCupvZG5ko_eiXAupbDfxWw",
    "description": "Welcome to CNNMoney's YouTube page. We cover everything from personal finance, autos and small business to the economy, markets and innovation.",
    "publishedAt": "2013-04-08T14:45:07.000Z",
    "resourceId": {
      "channelId": "UCe-4xQosMQGki8mT4sR98Q",
      "kind": "youtube#channel"
    },
    "thumbnails": {
      "default": {
        "url": "https://yt3.ggpht.com/-95dxd0EZ64Y/AAAAAAYAAA/AAAAAATA/VX7kX/AAAAAA/s88-c-k-no/photo.jpg"
      },
      "high": {
        "url": "https://yt3.ggpht.com/-95dxd0EZ64Y/AAAAAAYAAA/AAAAAATA/VX7kX/AAAAAA/s240-c-k-no/photo.jpg"
      }
    },
    "title": "CNNMoney"
  }
}
```
6.3.2 Google+ API

The Google+ API is the programming interface to Google+ that enables users to connect with each other for maximum engagement using Google+ features from within our application. Google uses the OAuth 2.0 protocol to allow authorized applications to access user data.

6.3.2.1 PeopleService

The PeopleService enables an application to get a person's profile and search through profiles. Each person has an uniquely identifying ID.

- GET /googleplus/PeopleService/search/{query}

Search all public profiles that match the API request parameters. Every request must specify an API key (with the key parameter).

**Path Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>type</th>
<th>required</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>string</td>
<td>required</td>
<td>Specify a query string for full text search of public text in all profiles.</td>
</tr>
</tbody>
</table>

**Query Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>type</th>
<th>required</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>required</td>
<td>Specifies an API key</td>
</tr>
<tr>
<td>language</td>
<td>string</td>
<td>optional</td>
<td>specifies the preferred language to search with.</td>
</tr>
<tr>
<td>maxResults</td>
<td>unsigned integer</td>
<td>optional</td>
<td>specifies the maximum number of people to include in the response, which is used for paging. For any response, the actual number returned might be less than the specified maxResults. Acceptable values are 1 to 50, inclusive. (default:25)</td>
</tr>
<tr>
<td>pageToken</td>
<td>string</td>
<td>optional</td>
<td>The continuation token, which is used to page through large result sets.</td>
</tr>
</tbody>
</table>

**Response Example**

This method returns a response with a structure similar to the following example:

```json
{
  "displayName": "EMERGENCY",
  "etag": "L2Xbn8bDuS5ErT6QA3PEQiwKYQxM/Jm9v3ntKacMsGG7br6DokxQjU_Q",
  "id": "109465517414289938955",
  "image": {
    "url": "https://lh6.googleusercontent.com/-eWGR6L4yPik/AAAAAAAABNI/AAAAAAAAABNI/CUhp84-kGIE/photo.jpg?sz=50"
  },
  "kind": "plus#person",
}  ```
- GET /googleplus/PeopleService/get/{userId}

Get a person's profile specified by the {userId} parameter. Every request must specify an API key (with the key parameter).

**PATH PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
<td>string</td>
<td>required</td>
<td>The ID of the person to get the profile for.</td>
</tr>
</tbody>
</table>

**QUERY PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>required</td>
<td>Specifies an API key</td>
</tr>
</tbody>
</table>

**RESPONSE EXAMPLE**

This method returns a response with a structure similar to the following example:

```json
{
  "aboutMe": "CNN International delivers up-to-the-minute information on the latest world, business, sports and entertainment headlines.",
  "circledByCount": 2757325,
  "cover": {
    "coverInfo": {
      "leftImageOffset": 0,
      "topImageOffset": 0
    },
    "coverPhoto": {
      "height": 528,
      "url": "https://lh6.googleusercontent.com/-m8wEfuaOnz8/U2tiqU_Vdsi/AAAAAAAAAOg/AloU08s5_Uc/s630-fcrop64=1,00003689bf87f59b/christiane.bg.twitter.cnn.jpg",
      "width": 940
    },
    "layout": "banner"
  },
  "displayName": "CNN International",
  "etag": "L2Xbn8bDuSErT6QA3PEQiwYKQxM/bHPgJ2_yrfpzW-Nkp957sMPTLtQ",
  "id": "11412484965716573853",
  "image": {
    "isDefault": false,
    "url": "https://lh6.googleusercontent.com/-m8wEfuaOnz8/U2tiqU_Vdsi/AAAAAAAAAOg/AloU08s5_Uc/s630-fcrop64=1,00003689bf87f59b/christiane.bg.twitter.cnn.jpg"
  }
}
```
6.3.2.2 ActivityService

The ActivityService enables an application to list a collection of activities, get an activity and search through activities.

- **GET /googleplus/ActivityService/list/{userId}**

List all of the activities in the specified collection for a particular user. Every request must specify an API key (with the key parameter).

**PATH PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
<td>string</td>
<td>required</td>
<td>Specifies the ID of the user to get activities for.</td>
</tr>
</tbody>
</table>

**QUERY PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>required</td>
<td>Specifies an API key</td>
</tr>
<tr>
<td>maxResults</td>
<td>unsigned integer</td>
<td>optional</td>
<td>specifies the maximum number of people to include in the response, which is used for paging. For any response, the actual number returned might be less than the specified maxResults. Acceptable values are 1 to 100, inclusive. (default:20)</td>
</tr>
<tr>
<td>pageToken</td>
<td>string</td>
<td>optional</td>
<td>The continuation token, which is used to page through large result sets.</td>
</tr>
</tbody>
</table>

**RESPONSE EXAMPLE**

This method returns a response with a structure similar to the following example:

```json
{
  "access": {
    "isPlusUser": true,
    "kind": "plus#person",
    "objectType": "page",
    "plusOneCount": 3019081,
    "tagline": "Go Beyond Borders",
    "url": "https://plus.google.com/+CNNInternational",
    "urls": [{
      "label": "http://edition.cnn.com/",
      "type": "website",
      "value": "http://edition.cnn.com/"
    }],
    "verified": true
  }
}
```
"description": "Public",
"items": [{
  "type": "public"
},
"kind": "plus#acl"
},
"displayName": "CNN International",
"id": "114124849657167573853",
"image": {
  "url": "https://lh6.googleusercontent.com/-Img8vmUFxc/AAAAAAAALAAA/AAAAAAQDY/WYVbthfadUA/photo.jpg?sz=50",
  "url": "https://plus.google.com/114124849657167573853",
  "etag": "\"L2Xbn8bDuSErT6QA3PEQiwYKQxM/oVEeNGn5kJw7Xcm3Z6UJ27vOU\"",
  "id": "z12xjrnyqnxVlVh23zxx0qfxvzrDhm04",
  "kind": "plus#activity",
  "object": {
    "attachments": [{
      "content": "To save the rhinos, one charity is moving them out of South Africa, where poaching is at an all time high.",
      "displayName": "Rhinos on a plane: Life-saving mission across borders",
      "fullImage": {
        "type": "image/jpeg",
        "url": "http://i2.cdn.turner.com/cnn/dam/assets/140911125226-rhinos-without-borders-sunset-story-top.jpg"
      }
    },
    "image": {
      "height": 303,
      "type": "image/jpeg",
      "url": "https://lh5.googleusercontent.com/proxy/kN4OyVdR03tyY1ev1pTcCzZPj9OeGWfYq4jTh5q4q4__HP8hSEQYH7M-YJxS1Fea_SL0Mz3Kags3BljZanjysYAUuDbpCuCodYvgbPAwY6isUp3oJ5QVj1ANkfPSSlasfHgz9tezxF57010pONFwu5nQ=w506-h303",
      "width": 506
    },
    "objectType": "article",
    "url": "http://cnn.it/1uxDNrK"
  }
}]}
"content": "Will these drastic measures stop poachers in South Africa? <a href="http://cnn.it/1uxDNrK">http://cnn.it/1uxDNrK</a> ",

"objectType": "note",

"plusoners": {
  "selfLink": "https://www.googleapis.com/plus/v1/activities/z12xjrnyqnxlxlvlv23zx0qfxvzzrdhm04/people/plusoners",
  "totalItems": 60
},

"replies": {
  "selfLink": "https://www.googleapis.com/plus/v1/activities/z12xjrnyqnxlxlvlv23zx0qfxvzzrdhm04/comments",
  "totalItems": 2
},

"resharers": {
  "selfLink": "https://www.googleapis.com/plus/v1/activities/z12xjrnyqnxlxlvlv23zx0qfxvzzrdhm04/people/resharers",
  "totalItems": 4
},

"url": "https://plus.google.com/114124849657167573853/posts/KKseYv2mU3j"
},

"provider": {
  "title": "Google+",
  "published": "2014-09-11T16:59:50.376Z",
  "title": "Will these drastic measures stop poachers in South Africa? http://cnn.it/1uxDNrK",
  "updated": "2014-09-11T16:59:50.376Z",
  "url": "https://plus.google.com/114124849657167573853/posts/KKseYv2mU3j",
  "verb": "post"
}

- GET /googleplus/ActivityService/get/{activityId}

Get the activity specified by the {activityId} parameter. Every request must specify an API key (with the key parameter).

**PATH PARAMETERS**

| activityId | string | required | Specifies the ID of the activity to get. |

**QUERY PARAMETERS**

| key | string | required | Specifies an API key |

**RESPONSE EXAMPLE**
This method returns a response with a structure similar to the following example:

```
{
    "access" : {
        "description" : "Public",
        "items" : [ {
            "type" : "public"
        } ],
        "kind" : "plus#acl"
    },
    "actor" : {
        "displayName" : "CNN International",
        "id" : "114124849657167573853",
        "image" : {
            "url" : "https://lh6.googleusercontent.com/-Img8vdmUFxc/AAAAAAAALAAAAA0QDY/WYVbthfadUA/photo.jpg?sz=50"
        },
        "url" : "https://plus.google.com/114124849657167573853"
    },
    "etag" : "\"L2Xbn8bDuSERt6QA3PEQiwYKQxM/oVEnNGn5kJw7XcolmSZ6UJ27vOY\"",
    "id" : "z12xjrnyqnxlxlvh23zzxqfxvzzrdhm04",
    "kind" : "plus#activity",
    "object" : {
        "content" : "To save the rhinos, one charity is moving them out of South Africa, where poaching is at an all time high."
    },
    "attachments" : [ {
        "displayName" : "Rhinos on a plane: Life-saving mission across borders",
        "fullImage" : {
            "type" : "image/jpeg",
        },
        "image" : {
            "height" : 303,
            "type" : "image/jpeg",
            "url" : "https://lh5.googleusercontent.com/proxy/kN4OyVdrR03tyY1evzZPj9OeGWQ=w506-h303",
            "width" : 506
        }
    },
    "objectType" : "article",
```
GET /googleplus/ActivityService/search/{query}

Search public activities. Every request must specify an API key (with the key parameter).

**PATH PARAMETERS**

<table>
<thead>
<tr>
<th>query</th>
<th>string</th>
<th>required</th>
<th>Full-text search query string.</th>
</tr>
</thead>
</table>

**QUERY PARAMETERS**

<table>
<thead>
<tr>
<th>key</th>
<th>string</th>
<th>required</th>
<th>Specifies an API key</th>
</tr>
</thead>
</table>
language string optional specifies the preferred language to search with.

<table>
<thead>
<tr>
<th>parameter</th>
<th>type</th>
<th>optional</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orderBy</td>
<td>string</td>
<td>optional</td>
<td>specifies how to order search results.</td>
</tr>
<tr>
<td>maxResults</td>
<td>unsigned integer</td>
<td>optional</td>
<td>specifies the maximum number of people to include in the response, which is used for paging. For any response, the actual number returned might be less than the specified maxResults. Acceptable values are 1 to 100, inclusive. (default:20)</td>
</tr>
<tr>
<td>pageToken</td>
<td>string</td>
<td>optional</td>
<td>The continuation token, which is used to page through large result sets.</td>
</tr>
</tbody>
</table>

**RESPONSE EXAMPLE**

This method returns a response with a structure similar to the following example:

```json
{
   "access": {
      "description": "Public",
      "items": [
         {
            "type": "public"
         }
      ],
      "kind": "plus#acl"
   },
   "actor": {
      "displayName": "College of Emergency Medicine",
      "id": "102515576988219247543",
      "image": {
         "url": "https://lh4.googleusercontent.com/-NtBkJGtt778/AAAAAAAAAAI/AAAAAAAAACc/sKjmbyLj7XQ/photo.jpg?sz=50"
      },
      "url": "https://plus.google.com/102515576988219247543"
   },
   "etag": "L2Xbn8bDuSErT6QA3PEQiwYKQxM/WBos1a0kWx8iX5Imj-FtiX9G9PQ\"",
   "id": "z12adm5ms2jirvqy04cizxi0x2ghvwhqg0k",
   "kind": "plus#activity",
   "object": {
      "attachments": [
         {
            "content": "College of Emergency Medicine Annual Scientific Conference 2014\n\nDr Gavin Lloyd - Sedation for the teensies: Ketamine/Propofol/Ketofol - the holy trinity and more\n\nConsultant in Emergency Medicine, Royal Devon & Exeter Hospital",
            "displayName": "Dr Gavin Lloyd - Adult Procedural Sedation 2014",
         }
      ],
      "url": "https://lh4.googleusercontent.com/-NtBkJGtt778/AAAAAAAAAAI/AAAAAAAAACc/sKjmbyLj7XQ/photo.jpg?sz=50"
   }
}
```
"embed": {
    "type": "application/x-shockwave-flash",
    "url": "https://www.youtube.com/v/aJELKr2SmsY?version=3&autohide=1&autoplay=1&feature=autoshare-u"
},
"image": {
    "height": 379,
    "type": "image/jpeg",
    "url": "https://lh4.googleusercontent.com/proxy/RavTaMPbncol2xVVPYbqyWgwRILt9jdC9qw1ks-mqzFhorTO2SKXxyRfdlI524xrdsqtqj4_YtH7zjv711uNG=w506-h379-n",
    "width": 506
},
"objectType": "video",
"url": "https://www.youtube.com/watch?v=aJELKr2SmsY&feature=autoshare"
},
"content": "",
"objectType": "note",
"plusoners": {
    "selfLink": "https://www.googleapis.com/plus/v1/activities/z12advm5ms2jirvqy04cizxil0x2ghfvwhqg0k/people/plusoners",
    "totalItems": 0
},
"replies": {
    "selfLink": "https://www.googleapis.com/plus/v1/activities/z12advm5ms2jirvqy04cizxil0x2ghfvwhqg0k/comments",
    "totalItems": 0
},
"resharers": {
    "selfLink": "https://www.googleapis.com/plus/v1/activities/z12advm5ms2jirvqy04cizxil0x2ghfvwhqg0k/people/resharers",
    "totalItems": 0
},
"url": "https://plus.google.com/102515576988219247543/posts/eH5ZQqWU9Cf"
},
"provider": {
    "title": ""
},
"published": "2014-09-12T11:44:07.131Z",
"title": ""
6.3.2.3 **CommentService**

The CommentService enables the application to list a collection of comments and get a comment.

- **GET /googleplus/CommentService/list/{activityId}**

List all of the comments for any activity. Every request must specify an API key (with the key parameter).

### Path Parameters

<table>
<thead>
<tr>
<th>Path Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>activityId</code></td>
<td>string</td>
<td>required</td>
<td>Specifies the ID of the activity to get comments for.</td>
</tr>
</tbody>
</table>

### Query Parameters

<table>
<thead>
<tr>
<th>Query Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>key</code></td>
<td>string</td>
<td>required</td>
<td>Specifies an API key</td>
</tr>
<tr>
<td><code>maxResults</code></td>
<td>unsigned integer</td>
<td>optional</td>
<td>specifies the maximum number of people to include in the response, which is used for paging. For any response, the actual number returned might be less than the specified maxResults. Acceptable values are 1 to 500, inclusive. (default: 20)</td>
</tr>
<tr>
<td><code>pageToken</code></td>
<td>string</td>
<td>optional</td>
<td>The continuation token, which is used to page through large result sets.</td>
</tr>
<tr>
<td><code>sortOrder</code></td>
<td>string</td>
<td>optional</td>
<td>The order in which to sort the list of comments.</td>
</tr>
</tbody>
</table>

### Response Example

This method returns a response with a structure similar to the following example:

```json
{
    "actor": {
        "displayName": "CM Maxwell",
        "id": "106978665941066060663",
        "image": {
            "url": "https://plus.google.com/106978665941066060663",
            "etag": "\"L2Xbn8bDuSER76QA3PEQiWYKQxM/VOVDxb9EefhKUXcygrCVqOPOI\\\"",
            "id": "212xjmxqnxlivelh23zzx0qfxvzzrdhm04.1410455053192631",
            "inReplyTo": {
                ...
            }
        }
    }
}
```
GET /googleplus/CommentService/get/{commentId}

Get the comment specified by the {commentId} parameter. Every request must specify an API key (with the key parameter).

**Path Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commentId</td>
<td>string</td>
<td>true</td>
<td>Specifies the ID of the comment to get.</td>
</tr>
</tbody>
</table>

**Query Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>true</td>
<td>Specifies an API key</td>
</tr>
</tbody>
</table>

**Response Example**

This method returns a response with a structure similar to the following example:

```json
{
  "actor": {
    "displayName": "CM Maxwell",
    "id": "106978665941066060663",
    "image": {
      "url": "https://lh3.googleusercontent.com/-E6QUqZJRZTQ/XafATfmckjo/photo.jpg?sz=50"
    },
    "url": "https://plus.google.com/106978665941066060663"
  },
  "etag": "L2Xbn8bDuSeERf6QA3PEQiJwKQxX+/VOVDxb9EefhKUXcygCvQP0Oik"
}
```

---

D5.1: Identification of Social Network Providers and API Design, Version V1, PU

"id": "z12xjinryqnxlxlvh23zzx0qfxvzrdhm04",
"url": "https://plus.google.com/114124849657167573853/posts/KKseYv2mU3j" }
,"kind": "plus#comment",
"object": {
"content": "Saving the animals is the humanitarian way but .....What about the humans; first."
"objectType": "comment"
,"plusoners": {
"totalItems": 1
"selfLink": "https://www.googleapis.com/plus/v1/comments/lxvlvh23zzx0qfxvzrdhm04#1410455053192631",
"updated": "2014-09-11T17:04:13.192Z",
"verb": "post"
}
6.4 Tumblr (not implemented)

Like Twitter, Tumblr has two APIs - a REST-based API for retrieving information from publicly available blogs (http://www.tumblr.com/docs/en/api/v2) and a firehose API that is available from third party data providers such as GNIP and Datasift.

6.4.1 Tumblr REST API

This is a basic API primarily intended to allow programmatic updates to a user’s blog. The most useful endpoint for EmerGent’s purposes is probably the Tagged method which returns the 20 most recent posts with the specified tag. The post can be returned in text (plain text) or raw (as the user originally entered it, including HTML tags). The REST API does not have facilities for free text search on blog text or for retrieving information about a blog’s owner. There is a rate limit on the number of calls but no guidance is available as to what it is.

A typical response would be similar to the following:

```json
{
"meta": {
  "status": 200,
  "msg": "OK"
},
"response": {
  "blog": { ... },
  "posts": [
    {
      "blog_name": "citriccomics",
      "id": "3507845453",
      "post_url": "http://citriccomics.tumblr.com/post/3507845453",
      "type": "text",
      "date": "2011-02-25 20:27:00 GMT",
      "timestamp": 1298665620,
      "state": "published",
      "format": "html",
      "reblog_key": "b0baQtsl",
      "tags": [
        "tumblrize",
        "milky dog",
        "mini comic"
      ],
      "note_count": 14,
    }
  ]
}
```
"title": "Milky Dog",
"body": "<p><img src="http:\/\/media.tumblr.com\/
tumblr_lh6x8d7LBB1qa6gy3.jpg"/>\n\nMilky Dog was inspired by Naomi Gee wrote on twitter, I really liked the hash tag #milkydog and quickly came up with a little comic about it. You can (and should) follow Naomi on twitter <a href="http:\/\/twitter.com\/@ngun" target="_blank">@ngun</a> I'm on twitter as well <a href="http:\/\/twitter.com\/@welewairplanes" target="_blank">@welewairplanes</a>

Also, if you're a Reddit user (or even if you're not) I submitted this there, if you could up vote it I'd be super grateful just <a href="http:\/\/tinyurl.com\/5wj3tqz" target="_blank">CLICK HERE</a>

},
...
"total_posts": 3
}

6.4.2 Tumblr Firehose

The Tumblr Firehose can be accessed via GNIP’s PowerTrack and Firehose products. Results are expressed in GNIP’s activity streams format. Costs are as specified in section 2.

The DataSift API also allows access to the Tumblr Firehose and provides powerful filtering capabilities. Results are returned in a custom format¹. Costs are given as $0.20 per 1000 – this will be in addition to the processing charge for the filter (see section 2).

6.4.3 Conclusions

The Tumblr REST API is free and is easy to use but is extremely limited. The lack of ability to search and to identify and hence to judge the reliability of the author of a post is a serious issue. It is probably best used for confirmation when possible tags relating to an emergency have been identified.

The Tumblr Firehose would be extremely useful but is again cost-prohibitive.

6.5 Instagram (not implemented yet)

6.5.1 Real-time API

Instagram provides a real-time API that will push notifications of new photos that match certain criteria. Subscriptions can be based on a geographical location and radius or on a specified tag, but not both together. Details of the media, including the username of the posting user, can be retrieved via the REST API.

6.5.2 REST API

It is possible to search for media in a given geographical radius and time frame. Radius is limited to 5 km and date range can be at most 7 days. Authenticated calls are limited to 5000 per hour per token. Unauthenticated calls are limited to 5000 per hour per application.

6.5.3 Conclusions

Instagram could be a useful source and picture data could be particularly helpful to emergency services; however, the limit of 5000 requests per hour could be restrictive.
7 Conclusion and Outlook

The aim of this deliverable is to establish a technical basis infrastructure for the following technical EmerGent work packages regarding the access to social media services. Therefore we first had to select a subsection of social media services this API focuses on (currently: Facebook, Twitter, YouTube, Google+) and to design and implement technical API to these services. In order to archive this purpose the following tasks have been performed:

- We have reviewed related approaches for gathering social media spanning data
- We have designed the overall architecture of the API
- We implemented and tested the API

The quantitative access restrictions to the source platforms are a critical point. Here a reliable mechanism has to be created, which makes sure that the request limits will not be reached even at high loads. For this, in our view, two mechanisms are eligible: On the one hand, if existing, the payment model of the platforms can be made use of; on the other hand the restrictions can be avoided by registering multiple applications on the respective platforms and switching between them. The latter has proved successful for a previous project, which dealt with a microblogging platform.

Furthermore, the Social Media API described here can be enhanced with further social media services. As an example, the Facebook access within the API allows currently only the gathering of public messages. With the aid of user-generated tokens the amount of information can be increased by private messages; however, for this, data protection and ethical issues have to be considered. The success of this approach to overcome the qualitative access restrictions of the source platforms is dependent on the number of users who are willing to share their data.

Additionally, the deployment of a geocoder and of a named entity recognizer is to be examined. Particularly public Facebook messages are rarely tagged with geo data; enriching these with appropriate data could lead to substantially better results for the geo search. However, there is the danger that the deployment of such tools could distort the results, because, for instance, a distinction between posts of the search area and posts about this area would be necessary. As a next step we are implementing access to Instagram (Section 6.5) to allow the Social Media API also a gathering of photos.
8 References
