



SEVENTH FRAMEWORK PROGRAMME  
FP7-ICT-2009-6

BlogForever  
Grant agreement n°. 269963

---

## D4.8: Final BlogForever Platform

---

<b>Editor:</b>	<b>J. García Llopis, R. Jiménez Encinar</b>
<b>Revision:</b>	First Version
<b>Dissemination Level:</b>	Public
<b>Author(s):</b>	J. García Llopis, R. Jiménez Encinar, Ş. Postacı, A. Çınar, G. Gkotsis, M. Rynning, M. Gulliksen, E. Banos, N. Naziridis, P. Chatzikamaris, I. Trochidis
<b>Due date of deliverable:</b>	May 31, 2013
<b>Actual submission date:</b>	September 14, 2013
<b>Start date of the project:</b>	March 01, 2011
<b>Duration:</b>	30 months
<b>Lead beneficiary name:</b>	European Organization for Nuclear Research (CERN)

Abstract:

This report presents the integration of the two components, the final weblog Spider and the final weblog digital Repository, fully functional and communicating optimally with each other, resulting in the Final BlogForever Platform. The implementation activities carried out during the last period as well as a detailed documentation of both components are provided.

**Project co-funded by the European Commission within the Seventh  
Framework Programme (2007-2013)**

The **BlogForever** Consortium consists of:

Aristotle University of Thessaloniki (AUTH)	Greece
European Organization for Nuclear Research (CERN)	Switzerland
University of Glasgow (UG)	UK
The University of Warwick (UW)	UK
University of London (UL)	UK
Technische Universitat Berlin (TUB)	Germany
Cyberwatcher	Norway
SRDC Yazilim Arastrirmave Gelistrirmeve Danismanlik Ticaret Limited Sirketi (SRDC)	Turkey
Tero Ltd (Tero)	Greece
Mokono GMBH	Germany
Phaistos SA (Phaistos)	Greece
Altec Software Development S.A. (Altec)	Greece

## Revision History

Version	Description of Change	Author	Date
0.2	First partial draft	J. García Llopis, R. Jiménez Encinar	20/08/2013
0.5	First draft	J. García Llopis, R. Jiménez Encinar	26/08/2013
0.6	Second draft (Implementation descriptions addition)	J. García Llopis, R. Jiménez Encinar, Ş. Postacı, A. Çınar	27/08/2013
0.7	Third draft (Updates on Introduction, Conclusions, Future Work, added contributions from CW and UW)	J. García Llopis, R. Jiménez Encinar, G. Gkotsis, M. Rynning, M. Gulliksen	28/08/2013
0.8	Fourth draft (General updates)	J. García Llopis, R. Jiménez Encinar,	29/08/2013
1	First version (Submitted)	J. García Llopis, R. Jiménez Encinar,	31/08/2013
1.1	Revision (Description of the open source blog spider implementation)	E. Banos, N. Naziridis, P. Chatzika-maris, I. Trochidis,	11/09/2013

# Table of Contents

<b>ExecutiveSummary</b>	<b>1</b>
<b>1 Introduction</b>	<b>2</b>
1.1 Background . . . . .	3
<b>2 Communication Mechanisms</b>	<b>5</b>
2.1 Spider Communication Mechanisms . . . . .	5
2.2 Repository Communication Mechanisms . . . . .	7
2.2.1 Data transfer . . . . .	7
2.2.1.1 Passive crawler . . . . .	7
2.2.1.2 Active crawler . . . . .	8
2.2.2 Data insertion into the repository . . . . .	9
<b>3 Spider Repository Integration</b>	<b>11</b>
3.1 Repository-Spider Direction . . . . .	12
3.2 Spider-Repository Direction . . . . .	13
<b>4 Implementation Updates</b>	<b>24</b>
4.1 Spider Implementation Updates . . . . .	24
4.2 Open Source Spider Implementation . . . . .	24
4.2.1 Software Architecture . . . . .	25
4.2.2 Operation . . . . .	27
4.3 Repository Implementation Updates . . . . .	28
4.3.1 Newly implemented features . . . . .	29
4.3.2 Updated features . . . . .	36

---

<b>5</b>	<b>Future Work</b>	<b>60</b>
<b>6</b>	<b>Conclusions</b>	<b>63</b>
	<b>References</b>	<b>64</b>
<b>A</b>	<b>Final Repository Implementation Descriptions</b>	<b>66</b>
A.1	Features already in Invenio . . . . .	67
A.2	List of implementation descriptions . . . . .	68
A.3	Features not retained . . . . .	143
<b>B</b>	<b>Final Spider Implementation Descriptions</b>	<b>144</b>
B.1	List of implementation descriptions . . . . .	145
B.2	Features not retained . . . . .	166
<b>C</b>	<b>Spider API Documentation</b>	<b>168</b>
<b>D</b>	<b>Deployment Instructions</b>	<b>174</b>
D.1	Repository Deployment Instructions . . . . .	174
D.1.1	Software Requirements . . . . .	174
D.1.2	Instructions . . . . .	174
D.2	Spider Deployment Instructions . . . . .	178
D.2.1	Software Requirements . . . . .	178
D.2.2	Instructions . . . . .	178

# List of Figures

1.1	BlogForever platform overview . . . . .	2
2.1	Common front-end, separate spiders . . . . .	6
2.2	Common front-end, some shared spider . . . . .	6
2.3	Passive crawler diagram . . . . .	8
2.4	Active crawler diagram . . . . .	9
2.5	Ingestion workflow diagram . . . . .	9
3.1	Ingestion workflow . . . . .	12
3.2	MARC extraction . . . . .	15
3.3	Cleaning HTML process . . . . .	16
3.4	Record and submission ID addition . . . . .	17
3.5	Attached files addition . . . . .	18
3.6	Parent blog record ID addition . . . . .	19
3.7	Parent blog visibility addition . . . . .	20
3.8	Parent blog topics addition . . . . .	21
3.9	Language addition . . . . .	22
3.10	Ingestion of original METS file in mongoDB database . . . . .	23

# List of Tables

4.1	Implementation Description: RF14 . . . . .	30
4.2	Implementation Description: RF34 . . . . .	30
4.3	Implementation Description: RF36 . . . . .	32
4.4	Implementation Description: RF46 . . . . .	34
4.5	Implementation Description: RF78 . . . . .	35
4.6	Implementation Description: RF87 . . . . .	35
4.7	Implementation Description: RF89 . . . . .	36
4.8	Implementation Description: RF1 . . . . .	38
4.9	Implementation Description: RF2 . . . . .	42
4.10	Implementation Description: RF3 . . . . .	43
4.11	Implementation Description: RF29 . . . . .	44
4.12	Implementation Description: RF52 . . . . .	47
4.13	Implementation Description: RF62 . . . . .	54
4.14	Implementation Description: RF65 . . . . .	56
4.15	Implementation Description: RF73 . . . . .	59
A.1	Implementation Description: RFID . . . . .	66
A.2	Implementation Description: RF1 . . . . .	69
A.3	Implementation Description: RF2 . . . . .	73
A.4	Implementation Description: RF3 . . . . .	74
A.5	Implementation Description: RF4 . . . . .	75
A.6	Implementation Description: RF6 . . . . .	77
A.7	Implementation Description: RF9 . . . . .	78
A.8	Implementation Description: RF12 . . . . .	79

A.9	Implementation Description: RF14 . . . . .	80
A.10	Implementation Description: RF17 . . . . .	80
A.11	Implementation Description: RF22 . . . . .	82
A.12	Implementation Description: RF23 . . . . .	82
A.13	Implementation Description: RF24 . . . . .	83
A.14	Implementation Description: RF25 . . . . .	84
A.15	Implementation Description: RF26 . . . . .	85
A.16	Implementation Description: RF28 . . . . .	86
A.17	Implementation Description: RF29 . . . . .	86
A.18	Implementation Description: RF31 . . . . .	90
A.19	Implementation Description: RF32 . . . . .	91
A.20	Implementation Description: RF34 . . . . .	92
A.21	Implementation Description: RF35 . . . . .	93
A.22	Implementation Description: RF36 . . . . .	94
A.23	Implementation Description: RF40 . . . . .	95
A.24	Implementation Description: RF41 . . . . .	96
A.25	Implementation Description: RF45 . . . . .	97
A.26	Implementation Description: RF46 . . . . .	98
A.27	Implementation Description: RF47 . . . . .	100
A.28	Implementation Description: RF48 . . . . .	101
A.29	Implementation Description: RF52 . . . . .	104
A.30	Implementation Description: RF53 . . . . .	105
A.31	Implementation Description: RF54 . . . . .	106
A.32	Implementation Description: RF57-58-61 . . . . .	107
A.33	Implementation Description: RF59 . . . . .	108
A.34	Implementation Description: RF62 . . . . .	115
A.35	Implementation Description: RF64 . . . . .	116
A.36	Implementation Description: RF65 . . . . .	118
A.37	Implementation Description: RF66 . . . . .	119



---

A.38	Implementation Description: RF67 . . . . .	120
A.39	Implementation Description: RF70 . . . . .	129
A.40	Implementation Description: RF71 . . . . .	134
A.41	Implementation Description: RF72 . . . . .	136
A.42	Implementation Description: RF73 . . . . .	139
A.43	Implementation Description: RF78 . . . . .	140
A.44	Implementation Description: RF87 . . . . .	141
A.45	Implementation Description: RF88 . . . . .	141
A.46	Implementation Description: RF89 . . . . .	142
B.1	Implementation Description: SFID . . . . .	144
B.2	Implementation Description: SF1 . . . . .	146
B.3	Implementation Description: SF2 . . . . .	147
B.4	Implementation Description: SF3 . . . . .	149
B.5	Implementation Description: SF4 . . . . .	150
B.6	Implementation Description: SF5 . . . . .	151
B.7	Implementation Description: SF6 . . . . .	152
B.8	Implementation Description: SF7 . . . . .	152
B.9	Implementation Description: SF8 . . . . .	153
B.10	Implementation Description: SF10 . . . . .	154
B.11	Implementation Description: SF11 . . . . .	156
B.12	Implementation Description: SF12 . . . . .	157
B.13	Implementation Description: SF13 . . . . .	157
B.14	Implementation Description: SF14 . . . . .	158
B.15	Implementation Description: SF15 . . . . .	159
B.16	Implementation Description: SF16 . . . . .	160
B.17	Implementation Description: SF17 . . . . .	161
B.18	Implementation Description: SF18 . . . . .	162
B.19	Implementation Description: SF19 . . . . .	163
B.20	Implementation Description: SF20 . . . . .	164

B.21 Implementation Description: SF21 . . . . . 165

B.22 Implementation Description: SF22 . . . . . 166

# Executive Summary

This document presents the work conducted for the integration of the final Weblog Digital Repository Component presented in *D4.7: Final Weblog Digital Repository Component*[1] and the final Weblog Spider Component presented in *D4.6: Final Weblog Spider Component*[2].

The communication capacities of each component are described in Chapter 2. Chapter 3 shows how those mechanisms have been used during the integration process which allow the communication between the Spider component and the Repository component. The latest implementation work done for each component is described in Chapter 4. Finally, Chapter 5 presents a set of suggestions to further extend the functionality of the BlogForever platform.

A demo of the latest Weblog Digital Repository version is available at:  
<https://blogforever.cern.ch>

# Chapter 1

## Introduction

The BlogForever project goal was to develop solutions for aggregating, preserving, managing and disseminating blogs. To achieve this goal, the BlogForever project aimed to develop a software platform that enables real-time harvesting and preservation of blog entities to facilitate extensive search and exploration functionalities of the archived blogs.

The BlogForever platform consists of two main software components: the Spider and the digital Repository. The Spider is responsible for crawling all the necessary blog data and characteristics designated for preservation while the Repository is responsible for long term archiving, preservation and management of the blogs, as well as providing facilities for further analysis and reuse of the content.



Figure 1.1: BlogForever platform overview

This deliverable intends to describe the integration between the two components: the weblog Spider and the digital Repository.

Finally note that, in the following text, the words *weblog* and *blog* will be used to describe the same concept, as well as *spider* and *crawler*.

## 1.1 Background

The BlogForever Description of Work (DoW) describes the objectives of the digital repository component as “being responsible for weblog data preservation. The digital repository will ensure weblog proliferation, safeguard their integrity, authenticity and long-term accessibility over time, and allow for better sharing and re-using of contained knowledge” [3].

Developing such a comprehensive archiving system from scratch is rather difficult and time consuming, and therefore avoided since there are many open-source archiving solutions. In this respect, the archiving system was selected as basis of the digital repository component was the Invenio<sup>1</sup> software suite developed at CERN<sup>2</sup>, which is also one of the partners involved in the BlogForever project.

In order to define how Invenio ought to be extended and modified, a list of requirements gathered from DoW, a weblog survey and 26 semi-structured interviews, were presented in *D4.1: User Requirements and Platform Specifications*[4]. Based on these requirements, 89 repository features, to be built on top of Invenio, were defined in *D4.4: Digital Repository Component Design*[5] as part of the design of the repository, in order to develop a complete digital repository for blogs. These features follow the metadata structure and preservation recommendations previously given by WP2 (*Weblog Structure and Semantics* in *D2.2: Weblog Data Model*[6]) and WP3 (*The BlogForever Policies* in *D3.1: Preservation Strategy Report*[7]).

The main goal of *Task 4.5: Implementation of the digital repository component* was to modify, to extend and to customize the vanilla Invenio source code by implementing the set of 89 repository features defined, and hence, to fulfill the BlogForever repository requirements.

In order to accomplish this goal, an initial prototype of the repository was delivered in *D4.5: Initial weblog digital repository prototype*[8], and the final Repository component was presented in *D4.7: Final Weblog Digital Repository Component*[1], which extends the initial prototype with the implementation of not only new features, but also updates to the existing ones.

Regarding the Spider, the BlogForever Description of Work (DoW) describes it as “capable of searching, harvesting and analysing large volumes of weblogs” [3]. The Spider design was presented in *D4.2: Weblog spider component design*[9] based on the input from D2.4, D2.5 and D2.6. A first prototype was presented in *D4.3: Initial Weblog Spider Prototype*[10] and the latest updates that the final version includes were described in *D4.6: Final Weblog Spider Component*[2].

The integration of the final Repository and the final Spider components is the outcome of the task *T4.6: Integration and Standardization*, as well as the purpose of this deliverable, resulting in the final BlogForever prototype platform.

Last but not least, implementation activities performed in WP4 are evaluated in WP5. To be more specific, implemented features are tested and validated during

---

<sup>1</sup><https://invenio-software.org/>

<sup>2</sup><http://www.cern.ch/>

*Task 5.2: Implementation of the case studies* based on the 6 case studies designed in *D5.1: Design and Specification of Case Studies*[11]. Since the implementation phase adopts an agile approach, testing phase adopts as well principles of agile testing which require testing to be an integral part of the software development. Moreover, feedback retrieved from both, internal and external testers, is used to improve the initial prototype towards the final digital repository component.

## Chapter 2

# Communication Mechanisms

Both components, the spider and the repository, have been designed in a way that avoids mutual dependency. They have communication mechanisms that let them work together but at the same time makes them flexible. In this Chapter the spider API and how any other repository could use it will be described. Also the repository's plugin mechanisms that let it communicate potentially with any spider.

How these communication mechanisms of both components have been used in the specific case of the BlogForever platform is detailed in Chapter 3.

## 2.1 Spider Communication Mechanisms

The Spider component is designed to have one front-end that serves as a management tool and a common spider communication endpoint, and have one or multiple spider installations. Because of this structure it is easy to scale up with more back-end when time comes. The communication between the front-end and the spider uses WCF<sup>1</sup> (Windows Communication Foundation) which is a service-oriented architecture principle to support distributed computing when services have remote consumers. Each service exposes its contract via one or more endpoints. An endpoint has an address and binding properties that specifies how the data will be transferred. It supports the following:

- Http binding
- Msmq binding
- Net Named Pipe binding
- Net Peer binding
- Net TCP binding
- UDP binding
- Web Http binding
- WSDual Http binding

---

<sup>1</sup><http://msdn.microsoft.com/en-us/library/ms731082.aspx>

- WS Http binding

There is no security mechanism attached to the communication between the front-end and the spiders. The same communication is used between the front-end and the world but with an extra security layer attached. These are the possible spider setups:

1. Common front-end, separate spiders

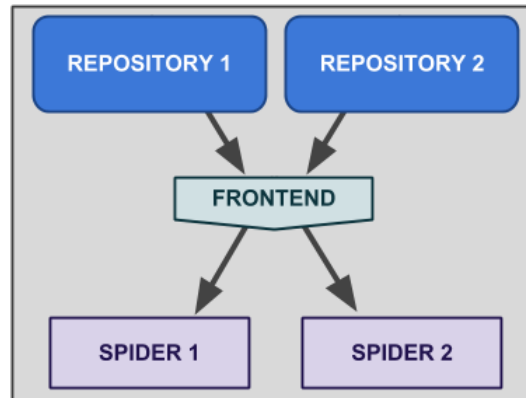


Figure 2.1: Common front-end, separate spiders

2. Common front-end, shared spiders

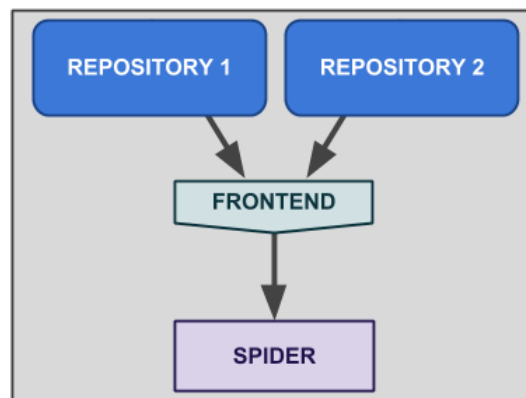


Figure 2.2: Common front-end, shared spider

The workflow of the spider is as following:

1. Startup
2. External client inserts one or more links
3. The system processes all unknown URLs and classifies them as



- (a) Blog
  - (b) Unknown
  - (c) Invalid
4. For each blog that has been classified create a rule set on how the given blog should be processed.
  5. For each blog that has been analyzed, perform extraction of content and store the result within a local search engine (Lucene<sup>2</sup>)
  6. Repeat the process.
  7. Client connects to the search engine and gets the latest crawled entities (blog/post/comment and binaries).

One of the methods is built in such a way that it converts the internal representation of the different objects (Blog, Post and Comment) into XML using XML serialization and XSLT transformation. The output from this method can be rewritten using the XSLT transformation to convert it to the appropriate format.

## 2.2 Repository Communication Mechanisms

The Repository component is very flexible regarding the communication with the crawler. It supports a system based on plugins that allow it to be customized in order to adapt it to any crawler. In the ingestion workflow we can differentiate 2 main stages: The first one is getting the crawler data to the repository servers (tagged as (1) in 2.3 and 2.4) It will be discussed in 2.2.1. The second one is inserting this data into the repository database (tagged as (2) in 2.3 and 2.4). It will be discussed in 2.2.2. For each of these 2 stages the repository supports the usage of custom plugins.

### 2.2.1 Data transfer

Depending on the mechanism chosen for the first stage, the crawlers can be categorized in 2 groups: passive crawlers and active crawlers. The repository is able to communicate with both kinds of crawlers.

#### 2.2.1.1 Passive crawler

The crawler offers some kind of querying mechanism that allows the repository to take control of the data download. The repository would be responsible of fetching the newly crawled data, download it to the repository servers and launch the ingestion task. In Figure 2.3, both (1) and (2) are blue because it is the repository who

---

<sup>2</sup><http://lucene.apache.org/>

controls both stages. Note that the fetcher is also part of the repository.

In this case, a fetcher component is needed. This fetcher would use the crawler's querying mechanisms to identify which files should be downloaded and will manage this download. The best way to implement such a fetcher in the repository would be using a *tasklet*. Tasklets are plugins that benefit from all the scheduling options that daemons have in Invenio. By writing a simple tasklet (Python code plugin) the developers are able to schedule the execution periodically, set the hours of the day when the tasklet is allowed to run, set its priority in the tasks queue and get a complete log of the tasklet execution, among other facilities.

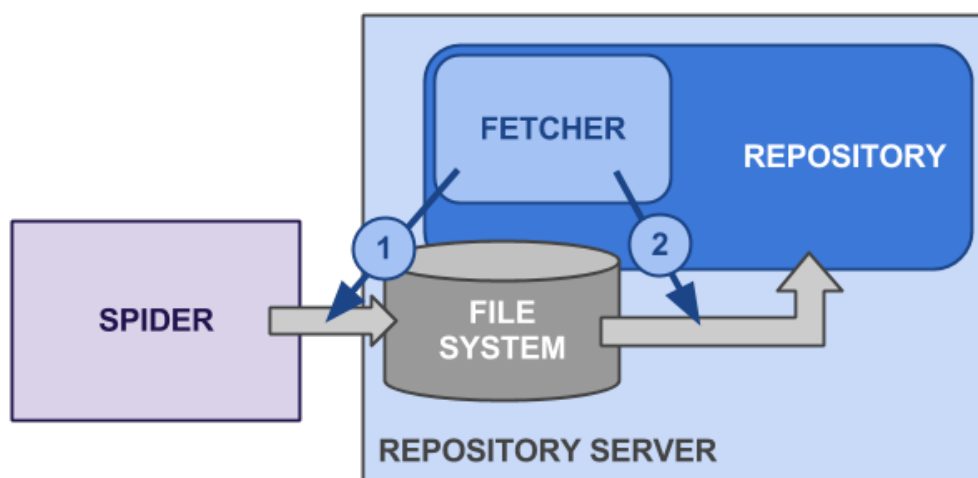


Figure 2.3: Passive crawler diagram

### 2.2.1.2 Active crawler

The crawler is able to push data to the repository servers (via FTP, for example). The repository would only be responsible of launching the ingestion task for the newly received data. Note that in Figure 2.4 (1) has the same color than the spider because it is the spider who pushes the crawled data.

In this case, the data transfer stage and its management are completely transparent to the repository. The administrators would need to configure the BatchUpload functionality of the repository. This tool, available in the Invenio code, checks periodically the file system and launches the BibUpload tasks. The BibUpload tasks will be introduced in the next subsection 2.2.2.

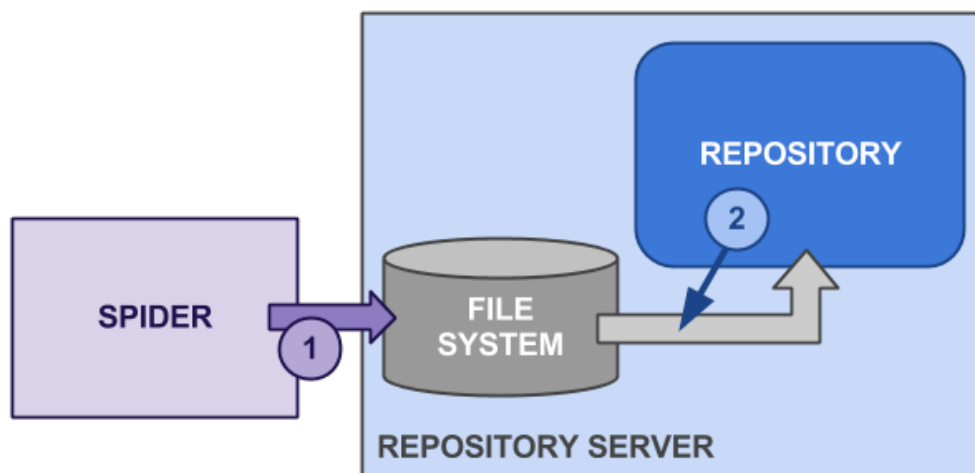


Figure 2.4: Active crawler diagram

## 2.2.2 Data insertion into the repository

The second stage inserts the data from the file system into the repository database and therefore creates Invenio-like records. The upload of records into Invenio is performed by BibUpload. Since the data coming from the fetcher may not be compatible with BibUpload, 2 different plugins have been added to the repository at this stage. They are of course optional and are part of the BibUpload task. Therefore, the workflow of the second stage of the ingestion, the data upload into the repository, results in the following 5 steps. The numeration corresponds to the numbers in Figure 2.5.

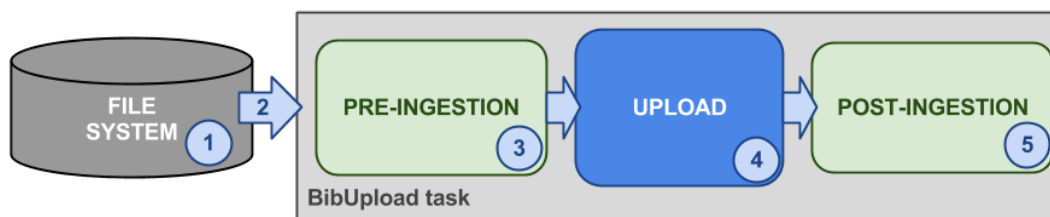


Figure 2.5: Ingestion workflow diagram

1. **File system:** Initially the files are stored in the file system, regardless how they arrived there during the first stage. It is recommended to use a multi-level directory structure for better scalability — the number of files to store for each record might be high and storing all the files in the same directory would slow down the process.
2. **BibUpload task submission:** The BibUpload task is submitted into the repository task queue scheduler: BibSched. The submission can be done by BatchUpload or by the fetcher itself. When calling to BibUpload, on top of

the rest of the options, 2 extra arguments can be passed. They are the names of the pre- and post-ingestion plugins to be used.

3. **Pre-ingestion:** This plugin can be used to transform the data and metadata coming from the crawler into an Invenio record that the repository can understand. Also, considering that any later modification of the metadata will result in a new BibUpload task being launched, it is recommended to do all the transformation and metadata enrichment before the upload. This will save many later BibUpload calls and will avoid overloading BibSched — the queue manager.
4. **Upload:** The record metadata is inserted into the repository database. Also the attached files are uploaded to the repository own file management system: BibDocFile.
5. **Post-ingestion:** This plugin can be used to perform tasks that only make sense if the upload has been successful. It can also be used to clean up the file system of downloaded files once they have been inserted into the repository.

## Chapter 3

# Spider Repository Integration

The purpose of *T4.6: Integration and Standardization*, is “to develop the BlogForever prototype platform, ensuring interoperability and comprising the final weblog Spider component and the final weblog digital Repository component, fully functional and communicating optimally with each other” [3].

The mechanisms defined with the aim of letting both components of the platform — the Spider and the Repository — communicate each other, were explained in detail in Chapter 2. Since this is clear, now is time to explain how these communication mechanisms have been used within the specific BlogForever’s use case.

The communication between the Spider and the Repository components was designed in the deliverable D4.4[5] to allow this process to occur in both directions: the Repository sends to the spider the new blog URLs submitted by users (repository-spider direction), and the Repository retrieves the newly crawled content by the Spider as well as ingests it into the repository database (spider-repository direction).

The ingestion workflow (represented in Figure 3.1) has as a starting point the Repository. Blog URLs can be inserted into the system through the submission web interface, and also by running a command-line tool (just available for administrators) that allows the submission of a batch of new URLs. Once the submission of a blog URL has taken place, the blog URL is sent to the Spider by using the spider web service API (1), so it can start crawling the content of the specific blog, and a new empty record is created and stored into a special hidden collection named Provisional Blogs (2). This collection contains all the empty blog records that have been created after the submission of the corresponding blog URLs.

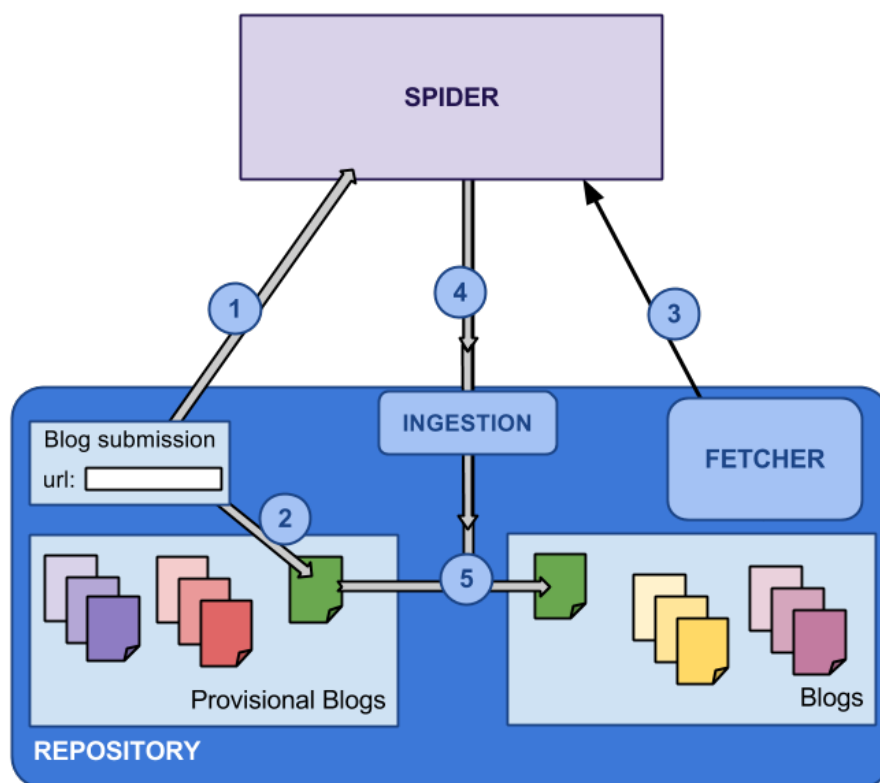


Figure 3.1: Ingestion workflow

In parallel with the submission procedure and the push action to the Spider, the Repository is periodically querying the Spider to check if there is any new content available to be downloaded (3). If so, once a blog URL has been processed by the Spider, the Repository fetches all the content associated to this blog URL (including the blog record itself)(4), the empty blog record that belongs to the Provisional Blogs collection and has the same URL is enriched with the metadata of the blog record retrieved from the Spider, and becomes part of the Blogs collection instead of the Provisional Blogs collection (5). Therefore, the Repository is also the component where the ingestion workflow ends.

### 3.1 Repository-Spider Direction

As mentioned in the introduction of this Chapter, the ingestion workflow has as a starting point the Repository. Blog URLs can be inserted into the system through both ways, the submission web interface and by running a command-line tool named BlogUploader (just available for administrators) that allows the submission of a batch of new URLs.

Once a blog URL is submitted through one of the two mechanisms mentioned above, the blog URL is directly sent to the Spider by using the spider web service API. The spider web service method that is invoked at this step is:

```
AddWatchPoints(xs:string apiKey, ns7:ArrayOfstring watchPoints)
```

The Repository will be constantly checking the status of the submitted blog URLs in the spider side. This will be explained in more detail in the following Section 3.2.

## 3.2 Spider-Repository Direction

As mentioned in Chapter 2, the crawlers can be categorized in 2 different groups: passive and active. For the BlogForever specific use case, the crawler is passive, being the repository the active component.

The ingestion process carried out by the Repository is done in 2 different phases. In the first phase, the content is fetched from the Spider component, while in the second phase, the retrieved content is inserted into the repository database.

In order to accomplish the objective of the first phase (to retrieve the content crawled by the Spider) a new tasklet called `bst_fetch_records_from_spider.py` has been implemented, which allows the Repository to fetch content from the Spider by using its web service API (see Appendix C for more details). Note that, in the following text, we would also refer to this tasklet as *fetcher*, following the terminology used in Chapter 2.

The parameters defined to be recognized by this tasklet are described as follows:

- `api_key`: This is the API key needed to connect to a specific Spider instance.
- `url`: This parameter corresponds to the url of the spider web service API we are accessing to.
- `constant_set`: This is the number of records that will be fetched and processed at once.
- `id_max`: This is the maximum record id to be fetched from the Spider which corresponds to the maximum value for a 32-bit signed integer (2147483647).

The steps that the fetcher follows to achieve its goal are:

1. To establish connection with the Spider using the API key provided and the corresponding web service URL. By default the tasklet will use the API key and the URL specified in the variables `CFG_SPIDER_API_KEY` and `CFG_SPIDER_WEBSERVICE_URL`.
2. To check the status of the submitted blog URLs in the spider side. The set of possible values is: *new* (URL still not processes), *unknown* (failed, we do not know what the URL is), *processing* (just wait), *invalid* (failed, page was not found), *entity* (success, WatchPoint created), *forward* (same as entity but the submitted URL was forwarded to another entity), *queued* (same as processing), *finished* (success), *failed* (temporarily not working, status will eventually change).

to entity, forward or invalid). The spider web service method that the fetcher invokes to get the status of the URLs is:

```
GetUriState(xs:string apiKey, xs:string uri)
```

If the status is *unknown*, *invalid* or *failed*, then the corresponding blog will be deleted from the Provisional Blogs collection since the Spider will never crawl this blog and the content will never come to the Repository. If the status is *forward*, the URL of the corresponding blog will be updated with the new one.

3. To retrieve the id of the last record that was fetched in the last execution of the tasklet in order to start downloading records from there.
4. To download the list of spider records crawled since the last execution of the fetcher. At this step the fetcher retrieves a set of records, for each record gets the METS file, as well as all its attached files (snapshot, css, html...), and validates the content of each of them. All the files are stored in temporary directories of the file system and the METS file is sent to the BibUpload task, which will carry out the transformation of the MARC embedded in the METS file into an Invenio-like MARC record, as well as the ingestion of the record into the repository database.

The spider web service methods that the fetcher invokes at this step are:

```
SearchEntities(xs:string apiKey, ns4:SearchRequest request)
GetDocumentAsMets(xs:string apiKey, xs:int documentId)
GetDocumentStorage(xs:string apiKey, xs:int documentId)
GetDocument(xs:string apiKey, xs:int documentId, xs:string filename)
```

As mentioned above, the fetcher invokes the BibUpload task so start uploading records to the repository database, taken as input the METS file retrieved from the Spider. This action corresponds to the second phase of the ingestion process.

In Chapter 2 it was mentioned that BibUpload has been extended with two new plugins with the objective of pre- and post-processing the records retrieved from the Spider to make them compatible with Invenio, as well as for preservation purposes. These two implemented plugins are described below.

The pre-ingestion plugin has been implemented as `bp_pre_ingestion.py`. The tasks performed at this stage are:

- Extracts the MARC from the METS received from the Spider. This MARC will be used as base to create an Invenio record into the Repository and it will be also enriched with new metadata as it is explained below.





Figure 3.2: MARC extraction

- The HTML included in MARC as text content is cleaned for 2 reasons: to remove tags that might affect the repository rendering style, and to scrape the remaining tags so they do not affect the validity of the document when it is exported to MARCXML.



Figure 3.3: Cleaning HTML process

- The record ID and the submission ID are added in the corresponding MARC tags. First, the code checks whether the record already exists in the Repository. In this case, the existing record ID will be added and the rest of the process will know that this is an update of an existing record. If it does not exist, a new record ID is created. The submission ID is the identifier used by the Spider.



Figure 3.4: Record and submission ID addition

- The attached files are also included in the Invenio record. With this purpose, FFT<sup>1</sup> tags are used to attach not just the files fetched from the Spider (images, etc.) but also the original METS file, and the HTML and CSS files crawled from the original web page.

<sup>1</sup><http://invenio-demo.cern.ch/help/admin/bibupload-admin-guide#3.6>



Figure 3.5: Attached files addition

- In the case of Posts and Comments, they have a blog parent or a post parent record, respectively. The record ID of the parent blog or parent post is located and used to populate the corresponding MARC tag. This makes much simpler and quicker to perform usual tasks in the Repository, like retrieving the list of all the Posts of a Blog, or all the Comments of a Post.



Figure 3.6: Parent blog record ID addition

- At submission time, the user decides the visibility of the blog (public, restricted, private). It is at pre-ingestion time when Posts and Comments inherit the visibility of the parent blog.



Figure 3.7: Parent blog visibility addition

- At submission time, the user selects which topic/s the blog belongs to. It is at pre-ingestion time when Posts and Comments inherit the topic/s of the parent blog.



Figure 3.8: Parent blog topics addition

- The language in which the records are written is not provided by the Spider so this is the moment, at pre-ingestion time, to detect the language of the records and to add it to the MARC in the corresponding tag. The language is needed in order to be able to translate the content of the record to a different language.



Figure 3.9: Language addition

Once the record is ready, it is inserted into the repository database by BibUpload. If this action ends with success, the post-ingestion plugin will be executed and will store the original METS file into the mongoDB for preservation purposes. It has been implemented as the BibUpload plugin named `bp_post_ingestion.py`. The tasks performed at this stage are:

- Gets the original METS file as it comes from the pre-ingestion plugin.
- Creates an XML tree from the METS file and extracts the record type (Blog, Post, Comment, Page).
- Identifies the Invenio record that corresponds to the METS file.
- The BibIngest module stores the original METS file in the submission database (mongoDB) using the record ID as identifier.



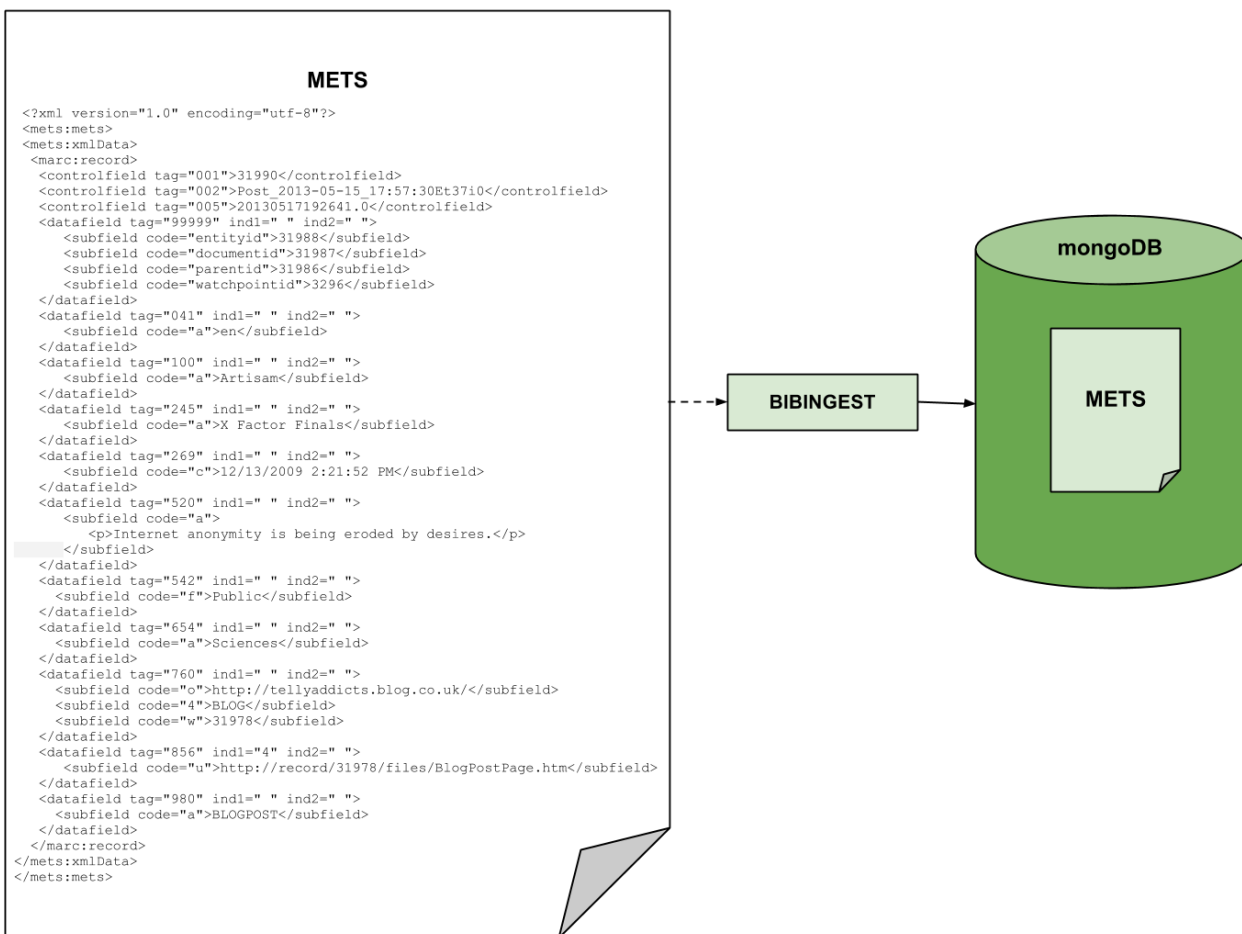


Figure 3.10: Ingestion of original METS file in mongoDB database

## Chapter 4

# Implementation Updates

During the last period of the project — since the submission of D4.6[2] and D4.7[1] — the development tasks have continued. This Section presents the implementation descriptions of the newly implemented features and the ones that have been fixed/updated during the last period.

The BlogForever project has used an iterative development for WP4 development processes. According to the DoW: “The design of Task 4.4 will be implemented through iterations. During these iterations, a new modification or add-on will be implemented, tested and documented each time” [3]. This continuous feedback from WP5 has been used to improve the stability and usability of the platform. WP4 has taken into consideration every suggestion and bug report coming from WP5 and addressed them one-by-one. All the bugs reported have been marked as solved in the bug tracking system used to communicate WP4 and WP5 and most of the usability suggestions have been implemented and are now part of the user interface.

### 4.1 Spider Implementation Updates

The weblog Spider was finalized by 1st May 2013. This final component was documented in the deliverable D4.6 [2]. There has been no additional features included after this implementation.

The last remaining 4 months the Spider has been used for several case studies and run on separate servers for CERN and AUTH.

This operation has resulted in some minor debugging and investigations of more type of blogs.

### 4.2 Open Source Spider Implementation

In this section, we present the open source BlogForever spider software architecture and operation.

This software is developed in parallel with the proprietary BlogForever spider designed [9] and implemented [2] as part of the original project description of work [3]. This implementation was necessary for two reasons. First, to be able to implement the BlogForever Business Exploitation Plan [12], which includes the provision of support services for the BlogForever downloadable open source software. Second, to experiment with the Spider - Repository Integration possibilities [5] and the development of multiple spiders which communicate in the same standard way with the BlogForever repository component.

In general, the features of the open source spider are implemented according to the BlogForever spider design as specified at D4.2: Weblog spider component design [9]. The difference between the open source spider and the proprietary spider is that the former uses python and open source technologies whereas the latter uses the proprietary .NET framework.

The first version of the open source weblog spider was finalised by the end of August 2013, but its development continues beyond the context of the BlogForever project.

### 4.2.1 Software Architecture

The development and operation of the open source spider was performed using debian linux 6. The software requirements of the open source spider are:

- *Python 2.6*: The version of python available in debian stable. It is also the version of python used in the BlogForever Repository.
- *python-lxml*: Standard python library to manipulate XML and HTML documents.
- *python-greenlet*: Python library necessary to implement lightweight in-process concurrent programming. This is necessary for the architecture of the spider.
- *libevent-dev*: An event notification library to execute callback functions when specific events occur on a file descriptor or after a timeout has been reached.
- *python-mysqldb*: Standard python library to implement mysql communication.
- *virtualenv*: A tool to implement isolated development environments. This is necessary to handle dependencies and versions, and indirectly permissions, during development and deployment.
- *Mysql 5+*: Database server
- *Redis 2.6+*: Nosql, advanced key-value store used for queue management.

The main modules of the open source spider are the following:

- *blog manager* module handles all blog related operations. Including talking to the database and coordinating all subsequent manager classes to process blogs.
- *blogpost manager* module contains all methods related to the processing of posts and post content. Those methods are exposed using a class called

blogpost\_manager which wraps around the complex process of downloading and caching the post content.

- *database manager* module contains the ORM classes that are used throughout the spider project to communicate with the database. It is based on SQLAlchemy.
- *entity metadata* is a module that handles the gathering of metadata from an entity page and local cache content. It accepts a dictionary of rules in the form of { metadata\_key\_name : XPath\_to\_value } where the XPath is a valid XPath string defining how can we get to the value given an entity's html markup code.
- *filesystem manager* is a module that manages all operations related to the local cache and filesystem.
- *logger factory* module is an auxiliary class that builds logging.logger objects using a uniform log format and handler. It also contains a runtime patch for the logging library, so that every logger built after the patching, can have new log methods. For the purposes of the BlogForever Spider, the new method implemented is the logger.update() method.
- *metadata manager* module handles the collection of metadata from the content of a specific entity. It makes use of a generic system that is described in rules.py. In short, it accepts a dictionary of XPath rules that are used to identify the blog engine. And after the engine becomes known, then rules, specific to the blog engine, are used to retrieve possible metadata.
- *ping server monitor* module monitors the pingservers provided and returns all the blogs both present in the fresh pingserver results and in the spider's database.
- *policy module* implements a configurable policy system for the update of the blogs in the BlogForever spider database.
- *rss autodetector* is a class that handles the detection of rss feed links given the HTML markup code of a blog. It uses feedparser, python's universal feed parser to detect if a given link is a valid feed or not.
- *rss manager* is a module that handles the detection, validation and updating of rss feed links, extracted from blog sites. The methods of the class are exposed by a class called rss\_manager.
- *rss update monitor* is the class that handles the monitoring of rss feeds. It will tell if a feed has been updated with a post or comment.
- *urlops* module is a collection of function related to the processing and validation of URLs and remote resources. It contains methods to normalize URLs and download their targets.
- *xml manager* module is responsible for building the necessary information for the implementation of the SOAP methods. It can build the responses for: SearchEntities, GetDocumentAsMets, GetFileStorageInfo and GetDocument API commands.

The file organisation of the software can be summarised as follows:

Folder structure:

- *spider/*: all the python source code. Note that in all source code files, there are docstrings describing them.
- *storage/*: all blog entities content is saved in this folder (tree structure created using sha1 of the entity url (basically, the entity id))
- *venv/*: virtual environment
- *log/*: all log files

Executables:

- *launcher.sh* spawns the workers
- *soap.py* runs the SOAP server
- *update.py* runs the spider (updates all blogs)

Configuration files:

- *settings.py*: universal configuration file
- *rqworker\_configuration.py*: rqworkers are background workers running tasks, this configuration is used when they are invoked.
- *rules.py*: metadata extraction rules for entities

## 4.2.2 Operation

The basic steps to operate the software are the following:

1. Install all required software as described in the previous section. Debian linux is the preferred operating system.
2. Download and extract the open source spider from the blogforever.eu website <sup>1</sup>.
3. Create mysql database and account.
4. Edit configuration files (*settings.py* and *rqworker\_configuration.py*).
5. execute *launcher.sh*. This spawns the *rqworkers* and traps the SIGINT so you can shut them down in batch. In the next versions, the *launcher.sh* script will run everything from the SOAP service to the update script.
6. To add and remove blog(s), use the cli tool *blog\_manage.py*:
  - (a) *-[a]dd* : it accepts a list of blog urls separated by space and adds them to the spider blog list. Usage:  

```
./blog_manage.py -add http://url.1.com http://url.2.com ...  
cat bloglist.txt — ./blog_manage.py -add
```

---

<sup>1</sup><http://blogforever.eu>

- (b) `-[r]emove` : it accepts a list of blog urls and if found in the spider's blog list, it removes them. (Note: the blogs are assigned the status 'deleted', but are not removed from the database.
- 7. To update the blogs in the spider, you can run (or schedule) the `update.py` script which handles it.
- 8. To raise the SOAP service you can run the `soap.py` script. Note that the SOAP service is compliant with the BlogForever repository - spider communication protocol.

The internals of the weblog harvesting process can be summarised as follows:

- There are three different actions in harvesting
  1. New entity (blog, post, comment)
  2. Update entity (blog, post, comment)
  3. Download content (blog, post, comment)
- In general, data are stored in mysql tables and job queues are stored in Redis. Job queues are populated with any type of the aforementioned actions. Job queues are based on Python RQ.
- RQ workers monitor continuously the queues continuously and execute the jobs. As soon as a job is finished, the relevant database tables and files in storage are updated. More specifically:
  1. New Entity: This process involves the registration of new entities to process.
    - (a) A new blog is defined by the user (`add_blog` functions) or by the SOAP Service request (`Add blog methods`).
    - (b) A new post / comment is identified by parsing the blog feeds.
    - (c) As soon as a new entity is detected proceed to Action 3. Download content is invoked.
  2. Update entity: Check blog feeds to identify new items to be processed. If a new entity is detected, then Action 1. New Entity, is invoked. This Action is executed periodically via RQ-scheduler.
  3. Download content:
    - (a) Using `wget`, mirror all entity web content (html, css, js, images, etc.) and extract relevant metadata from downloaded files.
    - (b) Finally, generate the METS XML and store it in `mets_content` database table.

## 4.3 Repository Implementation Updates

This Section presents the latest development work done within the Repository, which includes the implementation descriptions of the newly implemented features as well as the updates done on the already implemented ones.

### 4.3.1 Newly implemented features

In Chapter 2 of D4.7[1] a set of postponed features was listed. All these 8 features have been implemented and their implementation descriptions can be found in this subsection.

While the other 7 are features that were postponed, the feature *RF87 - The archive transforms the SIPS received from the spider to AIPs* is a different case since its design was updated in D4.7[1] according to the feedback received from WP3. The current design and implementation of this feature fulfills the preservation recommendations coming from D3.1[7] and hence enhances the OAIS compliance of the Repository.

The list of the newly implemented features and their implementation descriptions can be found below:

- RF14 - Descriptive statistics are offered by records
- RF34 - The archive displays and suggests similar records to the user
- RF36 - The archive identifies and stores the topic of blogs and blog posts to let users navigate through the archive by topic
- RF46 - Users can create personal collections of their favourite blogs
- RF56 - The archive provides a journal view of the new blog
- RF78 - The archive displays content after filtering it with user preferences
- RF87 - The archive transforms the SIPS received from the spider to AIPs
- RF89 - The archive carries out the normalization and/or migration of the media attachments

<b>Feature ID</b>	RF14 (Repository Feature 14)
<b>Name</b>	Descriptive statistics are offered by record
<b>Effort Spent</b>	4 Days
<b>Modules Affected / Created</b>	WebSearch
<b>Description of the new feature</b>	
<p>Enabled the “statistics” tab in the detailed view of a record. The information offered is “People who viewed this page also viewed”, “People who downloaded files from this page also viewed” and “Download history graph”</p>	
<b>Implementation details</b>	
<p>The logging of file downloads (<code>rnkDOWNLOADS</code>) and page views (<code>rnkPAGEVIEWS</code>) have been enabled.</p>	

Using this information, the `record_usage.py` code is able to show “People who viewed this page also viewed”, “People who downloaded files from this page also viewed” and a “Download history graph” using the WebStat module.

<b>Implemented By</b>	CERN
-----------------------	------

Table 4.1: Implementation Description: RF14

<b>Feature ID</b>	RF34 (Repository Feature 34)
<b>Name</b>	The archive displays and suggests similar records to the user
<b>Effort Spent</b>	1 Week
<b>Modules Affected / Created</b>	WebSearch, WebInterface
<b>Description of the new feature</b>	
<p>Extended Invenio’s record similarity functionality to consider blog records as a special case. To determine the similarity of two blogs, the level of similarity between their posts is first calculated and then aggregated to come up with a score.</p>	
<b>Implementation details</b>	
<p>Added an function step before <code>blog_extend_bibrank</code> in <code>search_engine.py</code> called <code>blog_extend_bibrank</code>. Given one blog, it will get all the posts, get the similar posts for all of them, then aggregate the results to get the most similar posts for the blog in general, and finally group them by parent blog. The final outcome is a list of blogs similar to the blog given as an argument.</p>	
<b>Implemented By</b>	CERN

Table 4.2: Implementation Description: RF34

<b>Feature ID</b>	RF36 (Repository Feature 36)
<b>Name</b>	The archive identifies and stores the topic of blogs and blog posts to let users navigate through the archive by topic
<b>Effort Spent</b>	2 Days
<b>Modules Affected / Created</b>	WebSubmit, WebBlog



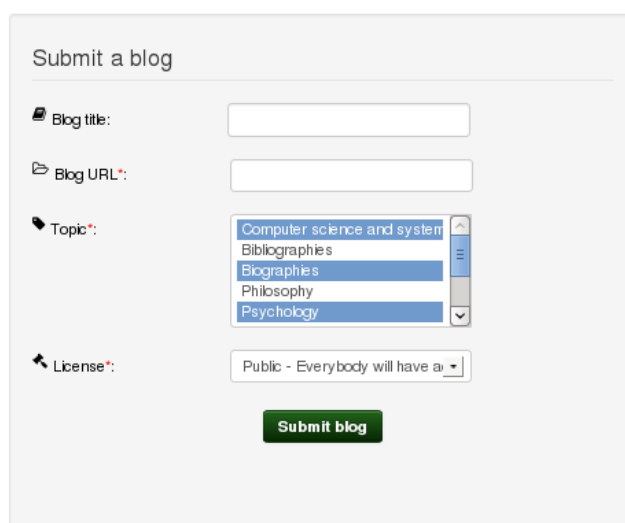
## Description of the new feature

At submission time users can select the topic/s the blog belongs to. After this, at pre-ingestion time, the selected topic/s are propagated to the descendants of the blog, so if users search for a specific topic, the search will return the blog and also all its descendants.

## Implementation details

A new config variable has been added named CFG\_BLOG\_TOPICS which define the list of topics that the repository offers to users. This list can be edit by the administrator.

A new field has been added to the submission form as a multiple select to be able to select more than one topic.



The image shows a web form titled "Submit a blog". It contains four input fields: "Blog title:" (text input), "Blog URL:" (text input), "Topic:" (multiple select dropdown menu), and "License:" (dropdown menu). The "Topic:" dropdown is open, showing a list of topics: "Computer science and system", "Bibliographies", "Biographies", "Philosophy", and "Psychology". The "License:" dropdown shows "Public - Everybody will have a...". A green "Submit blog" button is located at the bottom of the form.

The user can navigate through the topics by clicking on the drop menu displayed in the main page,

or also by clicking on the tags displayed with each record.

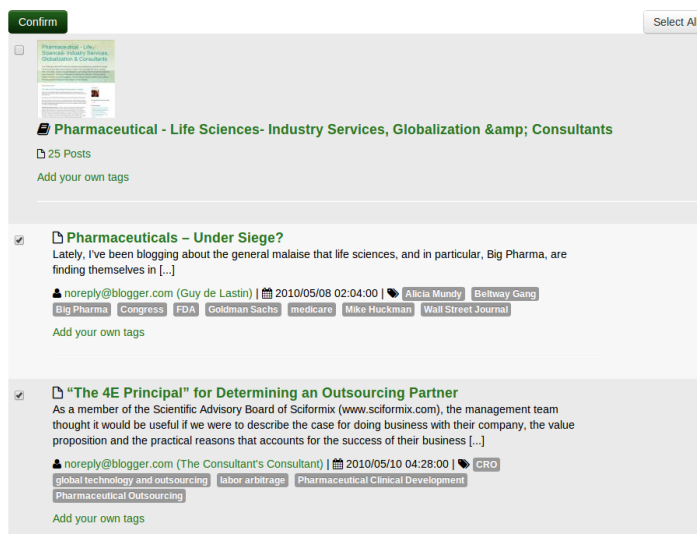
<b>Implemented By</b>	CERN
-----------------------	------

Table 4.3: Implementation Description: RF36

<b>Feature ID</b>	RF46 (Repository Feature 46)
<b>Name</b>	Users can create personal collections of their favourite blogs
<b>Effort Spent</b>	Days
<b>Modules Affected / Created</b>	WebBasket
<b>Description of the new feature</b>	
<p>With this feature blogs can be added to user personal baskets with their blog posts.</p>	

When deleting a blog from a personal basket, user is asked to select which blog posts of that blog s/he also wants to delete.

When deleting a blog post from a personal basket, checks if the blog of that blog post is also in the same basket. If so, user is asked to select which blog posts of that blog s/he also wants to delete.



## Implementation details

- `perform_request_add` function of `webbasket.py` is modified to add the blog posts of the blog when the user adds a blog to personal basket.
- `modify` method of `WebInterfaceYourBasketsPages` class is modified to check if there are any other records user may want to delete when removing a blog or blogpost from a personal basket.
- `perform_request_confirm_delete` is added to `webbasket.py` to display the list of related blogs and blog posts with the selected record.
- `tmpl_delete_related_record_confirmation` method is introduced in `Template` class of `WebBasket` module, which constructs the page containing the list of the records which the user may also want to remove.
- `confirm_delete` method is introduced in `WebInterfaceYourBasketsPages` class, which deletes the selected blogs and blog posts.
- `get_basket_recids` is added to `webbasket_dblayer.py` that returns the list of the record IDs of given basket.
- Two new functions are added into `webblog_utils.py` :
  - `extend_with_blog_posts` : Extends the list of blog IDs with their blog post IDs.

- [get\\_related\\_records\\_in\\_basket](#) : Returns the IDs of the blog and blog posts related with the given record from the given basket.

<b>Implemented By</b>	Alper Çınar (SRDC)
-----------------------	--------------------

Table 4.4: Implementation Description: RF46

<b>Feature ID</b>	RF78 (Repository Feature 78)
<b>Name</b>	The archive displays content after filtering it with user preferences
<b>Effort Spent</b>	2 Days
<b>Modules Affected / Created</b>	WebSearch
<p><b>Description of the new feature</b></p> <p>The repository offers to users the possibility of personalizing their searches with some options such as: the collections where users wish to search and the number of records to be displayed per page.</p> <p><b>Implementation details</b></p> <p>The search user preferences has been enriched with a multiselect where the user can select the collections that will be taken into account in his searches. Also the file <i>websearch_blueprint.py</i> has been amended in order to propagate the selected preferences.</p> <div style="text-align: center;"> <p><b>Edit</b></p> <hr/> <p>Results per page: <input type="text" value="25"/></p> <p>Hotkeys: <input type="text" value="Disable"/></p> <p>Collections: <input type="text" value="Next BlogForever Repository - Blogs Posts Comments"/></p> <hr/> <p><input type="button" value="Send"/></p> </div>	
<b>Implemented By</b>	CERN

Table 4.5: Implementation Description: RF78

<b>Feature ID</b>	RF87 (Repository Feature 87)
<b>Name</b>	The archive transforms the SIPS received from the spider to AIPS
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	BibArchive
<b>Description of the new feature</b>	
<p>New module to archive the AIP (Archival Information Package) of a record. A daemon checks periodically for new or modified records, creates an AIP and stores it in a dedicated database. In the web interface, the detailed view of a record offers the user a link to download the AIP.</p>	
<b>Implementation details</b>	
<p>A new module called BibArchive has been implemented: The database used is MongoDB. It supports versioning, but the old versions of an AIP can only be retrieved from the command-line interface. The AIP is wrapped using BagIt and then zipped. The zipped file is stored in the db directly. It includes:</p> <ul style="list-style-type: none"> <li>• The MARC and METS xml files of the record</li> <li>• All the attached files (mostly images)</li> <li>• The available metadata and md5 checksums of each file</li> </ul> <p>A new tasklet has been created to update the database. The administrator can decide how often this tasklet runs (depending on the upload/modification rates, a value between 30 mins and 1 day is recommended). When it runs, it first retrieves then list of records that have been created or modified since the last time it run. Then, for each them, an AIP is created using BagIt and the content mentioned above and inserts it in the archival database.</p>	
<b>Implemented By</b>	CERN

Table 4.6: Implementation Description: RF87

<b>Feature ID</b>	RF89 (Repository Feature 89)
<b>Name</b>	The archive carries out the normalization and/or migration of the media attachments
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	BibSched
<b>Description of the new feature</b>	
<p>New tasklet that creates new version of the file of a record, migrating them from one format to another. It accepts plugins to transform the files</p>	
<b>Implementation details</b>	
<p>A new plugin system has been implemented allowing the administrators to implement their own format conversion code and place the files under <code>modules/bibdocfile/lib/format_migration_plugins/</code> Then, the variable <code>CFG_FORMAT_MIGRATION_PLUGINS_MAPPING</code> has to be updated to reflect which plugin is desired to be used for each format.</p> <p>A new tasklet has been implemented to migrate the format of attached files. It retrieves the list of records uploaded or modified since the last time the tasklet run and applies the appropriated conversion plugin according to the variable <code>CFG_FORMAT_MIGRATION_PLUGINS_MAPPING</code>. Finally, it uploads the new file into the record as a new format of the same file, without replacing it.</p>	
<b>Implemented By</b>	CERN

Table 4.7: Implementation Description: RF89

### 4.3.2 Updated features

The following implementation descriptions describe the features that were implemented during previous development iterations, but have been modified since then because, they have just been fixed or improved, or because they have been ported to the Invenio *next* branch. These implementation descriptions should be considered as an update to the ones that can be found in Chapter 3 of D4.5[8] and Chapter 3 of D4.7[1].

- RF1 - Customizable user dashboard
- RF2 - “Your History” box as part of the user dashboard

- RF3 - “Share” option in “Your History” box
- RF29 - The archive alerts the user when there are software updates
- RF52 - Users can tag archived records with personal tags
- RF62 - Export records as PDF and JPEG
- RF65 - The archive analyzes blog links and stores the connections between them separately
- RF73 - The archive recommends blogs to users based on the ratings and preferences

<b>Feature ID</b>	RF1 (Repository Feature 1)
<b>Name</b>	Customizable user dashboard
<b>Effort Spent</b>	Days
<b>Modules Affected / Created</b>	WebSession
<b>Description of the new feature</b>	
<p>The layout of the dashboard is re-organized. Floating layout has become 3-column layout so that each column become independent from each other.</p> <p>Each box in user dashboard has become collapsible. They can be collapsed by clicking 'x' symbol on top right corner of the box.</p> <div data-bbox="547 801 1002 985" data-label="Image"> </div> <p>The list of the collapsed boxes takes place on top of the dashboard. A collapsed box can be retrieved back by clicking on its name.</p> <div data-bbox="459 1128 1091 1155" data-label="Image"> </div>	
<b>Implementation details</b>	
<ul style="list-style-type: none"> <li>• To change the layout of the user dashboard <code>webaccount_display.html</code> is modified and <code>webaccount_widget.html</code> is created.</li> <li>• The <code>index</code> endpoint of <code>webaccount_blueprint.py</code> is modified to display the dashboard boxes in proper column.</li> <li>• To expand the collapsed boxes in user dashboard, <code>loadwidget</code> endpoint is introduced in <code>webaccount_blueprint.py</code>.</li> </ul>	
<b>Implemented By</b>	Alper Çınar (SRDC)

Table 4.8: Implementation Description: RF1




<b>Feature ID</b>	RF2 (Repository Feature 2)
<b>Name</b>	“Your History” box as part of the user dashboard
<b>Effort Spent</b>	Weeks
<b>Modules Affected / Created</b>	WebSession

### Description of the new feature

“Your Activities” page is re-constructed while porting this feature to next. This page includes the user’s history in reverse chronological order with 15 main categories:

- Adding into basket
- Basket creation/subscription
- Notes on baskets
- Votes
- Page views
- Searches
- Subscriptions
- Downloads
- Messages
- Alerts
- Discussions
- Reviews
- Reports
- Groups
- Payments

 21/08/2013 10:43:25      You displayed [Das Bedeutet](#) > [Das Bedeutet](#).

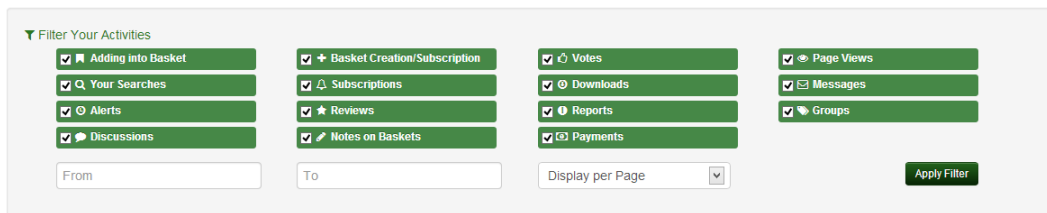
RF2 - A sample item on “Your Activities” page

By default, the most recent 10 activities are displayed on “Your Activities” page. User may load older activities through “Load older activities” link on the bottom of the page.

[Load Older Activities](#)

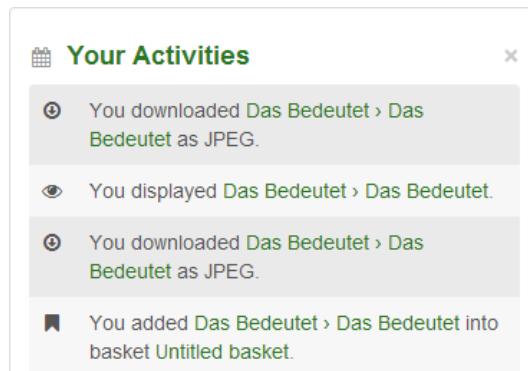
RF2 - The link to display older activities.

A filter is created to filter the categories that are displayed on “Your Activities” page.



RF2 - Filter panel

The new box for “Your Activities” is created in the user dashboard. It consists of last 10 activities of the user.



## Implementation details

- **HistoryElement** class is introduced to represent each history item in the “Your Activities” page. It has only one constructor and the following instance variables:
  - **date** : Date of the history element.
  - **entry** : Database entry corresponds to that history element.
  - **id** : ID of the entry.
  - **history\_type** : Type of the history.
  - **icon** : Icon that is displayed near the activity message on the web interface.
  - **msg** : Message that is displayed on the web interface.
- **HistoryCollector** abstract class is introduced to create History Collectors for each category. It has the following instance variables and methods:
  - **icon** : Icon for collected **HistoryElement** s.
  - **label** : Label to filter this collector.
  - **check** : Returns the most recently fetched **HistoryElement** s. (This method should not be overridden in its child classes)

- **use** : Fills the required attributes of the most recently fetched **HistoryElement** and returns it. (This method should not be overridden in its child classes)
  - **get\_user\_history** : Fetches and returns entries from database.
  - **get\_id** (static): Returns the ID of the given entry.
  - **get\_date** (static): Returns the date of the given entry.
  - **get\_message** (static): Returns the message displayed on web interface.
- 15 new classes are inherited from **HistoryCollector** class to collect user history. These are:
    - **PaymentHistory**
    - **AlertHistory**
    - **BasketCreationHistory**
    - **BasketNoteHistory**
    - **BasketAddHistory**
    - **DiscussionHistory**
    - **ReportHistory**
    - **ReviewHistory**
    - **SubscriptionHistory**
    - **VoteHistory**
    - **MessageHistory**
    - **DownloadHistory**
    - **SearchHistory**
    - **RecordViewHistory**
    - **GroupHistory**
- **HistoryManager** class introduced to collect the user history from registered **HistoryCollector** s. It has the following methods:
    - **get\_user\_history** : Fetches and returns the list of the user history.
    - **has\_mote\_history** : Returns if the user has more LHistoryElements to fetch.
    - **get\_user\_history\_json** : Returns the user history as JSON format.
- **webhistory\_index.html** is introduced to display “Your Activities” page.
  - **webhistory\_blueprint.py** is introduced and it has the following endpoints:
    - **index** : Constructs the “Your Activities” page.
    - **filteractivities** : Filter the activities according to the filter panel of the user interface.
    - **getmore** : Returns the most recent activities that are not displayed in “Your Activities” page.


- **webhistory\_user\_settings.py** is introduced to add “Your Activities” box to the user dashboard.
- **webhistory\_config.py** is introduced. It keeps the following configurations:
  - **CFG\_WEBHISTORY\_MSGS** : Keeps the messages to display user activities.
  - **CFG\_WEBHISTORY\_DATEPICKER\_DATETIME** : Format of the date picker of the filter in “Your Activities” page.
  - **CFG\_WEBHISTORY\_JSON\_DATE\_FORMAT** : Date format when using the dates of the activity in JSON objects.
  - **CFG\_WEBHISTORY\_JSON\_TIME\_FORMAT** : Time format when using the dates of the activity in JSON objects.
  - **CFG\_WEBHISTORY\_JSON\_DATETIME\_FORMAT** : Concatenation of the date format and time format.
- **CFG\_WEBHISTORY\_DATETIME\_FORMAT** , which keeps the format of the datetime displayed in “Your Activities” page and box, is added to **invenio.conf**
- The type of the **date\_creation** column of **UserQueryBasket** is changed from **Date** to **DateTime** .
- Two new columns **creation\_date** and **action\_code** are added into **UserBskBASKET** table. **webbasket\_dblayer.py** is updated to accommodate these changes.

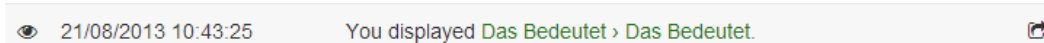
<b>Implemented By</b>	Şenan Postacı(SRDC)
-----------------------	---------------------

Table 4.9: Implementation Description: RF2

<b>Feature ID</b>	RF3 (Repository Feature 3)
<b>Name</b>	“Share” option in “Your History” box
<b>Effort Spent</b>	Days
<b>Modules Affected / Created</b>	WebSession, WebMessage

#### Description of the new feature

To share an activity, a tiny  icon is added near each activity in the “Your Activities” pages.



RF3 - A sample item on “Your Activities” page with sharing option

By clicking this icon, users may share their activities through a modal window. This modal window contains a form to send the activity to users.

### Implementation details

- **serialized** instance variable is added to **HistoryElement** to generate messages for shared activities.
- New methods are added to **HistoryCollector** to create and share user activities:
  - **serialize** (static): Serializes given database entry as list to use it for constructing message of the shared activity.
  - **get\_share\_message** (static): Converts serialized database entry to message.
- **share** endpoint is introduced in **webmessage\_blueprint.py** to perform sharing an activity.
- **view** endpoint of **webmessage\_blueprint.py** and **webmessage\_view.html** are updated to display messages of shared activities.
- **getsharemessage** endpoint is introduced in **webhistory\_blueprint.py** to get the message of given shared activity.
- Scripts of the **webmessage\_add.html** is moved to **webmessage\_common\_js.html** to reuse them in **webhistory\_index.html** while sharing an activity message.
- **hstSHARE** table is introduced to keep the shared activities. **webmessage\_query.py** is also updated accordingly.
- **CFG\_SHARE\_MSGS** , which keeps the messages for shared activities, is added into **webhistory\_config.py** .

**Implemented By** | Şenan Postacı(SRDC)

Table 4.10: Implementation Description: RF3

<b>Feature ID</b>	RF29 (Repository Feature 29)
<b>Name</b>	The archive alerts the user when there are software updates
<b>Effort Spent</b>	Days
<b>Modules Affected / Created</b>	WebSession
<b>Description of the new feature</b>	
<p>The archive checks and informs the administrators via user dashboard when there are software updates.</p> <p>A newer version of Invenio is available for download. You may want to visit <a href="http://invenio-software.org/wiki/Installation/Download">http://invenio-software.org/wiki/Installation/Download</a></p> <p>RF29 - The alert displayed on the user dashboard when there are software updates</p>	
<b>Implementation details</b>	
<ul style="list-style-type: none"> <li>The <a href="#">index</a> endpoint of <a href="#">webaccount_blueprint.py</a> is modified to check if there are any software updates.</li> </ul>	
<b>Implemented By</b>	Alper Çınar (SRDC)

Table 4.11: Implementation Description: RF29

<b>Feature ID</b>	RF52 (Repository Feature 52)
<b>Name</b>	Users can tag archived records with personal tags
<b>Effort Spent</b>	3 Weeks
<b>Modules Affected / Created</b>	WebTag
<b>Description of the new feature</b>	
<p>Users are able to assign personal tags to the records. With these tags, the user is able to organize the records in personal collections according to their preferences.</p> <p>Users are able to create and delete tags, as well as attach and detach them from records.</p>	

## Implementation details

The functionality is provided by a new module called WebTag. The module utilizes infrastructure provided by Invenio Next.

### Database Backend

WebTag defines a set of classes which are translated into database tables using SQL Alchemy ORM:

- **WtgTAG** : A named tag object owned by a user. The tag's name cannot contain non-alphanumeric characters.
- **WtgTAGRecord** : Association between a WtgTAG and a record (class **Bibrec** ).
- **WtgTAGUsergroup** : Describes the permissions given by the owner to a group of users. This mechanism is not yet implemented, but we plan adding group and public tags later.

### User Interface

WebTag defines an **InvenioBlueprint** . The user interface consists of:

- Tag Cloud mode of viewing tags (</yourtags/display/cloud>)

#### Tag Cloud



asshole fff fun interesting Qua Quo  
to read to share work

- Table mode of viewing tags (</yourtags/display/list>): this view allows users to create and delete tags as well as display them ordered by different parameters.

## Your tags

### Tag list

<input type="checkbox"/>	Tag	Documents	Action
<input type="checkbox"/>	asshole	1	Delete
<input type="checkbox"/>	fff	1	Delete
<input type="checkbox"/>	fun	1	Delete
<input type="checkbox"/>	interesting	3	Delete
<input type="checkbox"/>	Qua	0	Delete
<input type="checkbox"/>	Quo	0	Delete
<input type="checkbox"/>	to read	3	Delete
<input type="checkbox"/>	to share	1	Delete
<input type="checkbox"/>	work	0	Delete

Delete Selected

New tag

- List of tags in document search and detailed view: for logged-in users, a list of tags is displayed when viewing a document.



- Tag editor: in document view, users can open the editor to attach and detach tags from a record. It can also create new tags and attach them. The editor uses javascript and commits all changes immediately to the server by AJAX requests.
- List of records associated with a tag ([/yourtags/tag/<id\\_tag>/records](/yourtags/tag/<id_tag>/records)): Displays all records associated with the selected tag.
- User Settings widget in Your Account page presents statistics about tags as well as links to other parts of the module's user interface.

All information sent to the server is validated using WTForms validators in order to maintain name restrictions and check users' permissions.

The list of tags in document view is implemented with the template context function `bfm_webtag_record_tags`. The function can be used in various templates which allows easy customization of application interface.

### Configuration

The configuration variables are located inside `webtag-config.py`:



- **CFG\_WEBTAG\_NAME\_MAX\_LENGTH** : maximum length of tag name.
- **CFG\_WEBTAG\_NAME\_REPLACEMENTS\_SILENT** : list of regular expression replacements applied before saving a tag name.
- **CFG\_WEBTAG\_NAME\_REPLACEMENTS\_BLOCKING** : list of regular expression replacements applied on a tag name. If a match is found, the name is considered invalid. The expressions are used to suggest a similar valid name.
- **CFG\_WEBTAG\_ACCESS\_NAMES** and **CFG\_WEBTAG\_ACCESS\_RIGHTS** : permission levels for group and public tags. Currently not used.

### File structure

The module's files:

- **lib/\*** : Application logic in Python and Javascript.
- **etc/templates/\*** : Jinja2 templates for generating HTMLs. Files without **\_base.html** suffix are provided to simplify interface customizations.

<b>Implemented By</b>	CERN
-----------------------	------

Table 4.12: Implementation Description: RF52

<b>Feature ID</b>	RF62 (Repository Feature 62)
<b>Name</b>	Export records as PDF and JPEG
<b>Effort Spent</b>	Months
<b>Modules Affected / Created</b>	BibFormat
<b>Description of the new feature</b>	
<p>PDF and JPEG are added as new export options:</p> <div data-bbox="549 1724 1003 1890" data-label="Image"> </div> <p>PDFs are created by LaTeX templates. First, the record is converted from HTML to LaTeX, then it is converted from LaTeX to PDF.</p>	


## AstroDAbis

Archived date: 2013/08/22

Original Blog URL: <http://astrodabis.jiscinvolve.org/wp>

### Posts in this blog

#### AstroDAbis 's beta service

 normangray | Posted Date: 2012/03/22 Archived date: 2013/08/22


The AstroDAbis project 's annotation service is now up and running at [astrodabis.roe.ac.uk](http://astrodabis.roe.ac.uk) , for your delight and delectation.

This is a beta service. It works, but we think we still have some work to do to make it a bit more user-friendly, and to integrate it more fully into the set of services that ROE offers. That 's scheduled to happen over the summer, as part of ROE 's intended revamp of its services.

If you spot any problems with the service, please do let us know.

This entry was posted on Thursday, March 22nd, 2012 at 12:33 and is filed under [Uncategorized](#) .

#### Moving gracefully towards the end...

 normangray | Posted Date: 2012/02/17 Archived date: 2013/08/22

The AstroDAbis project is moving towards its final stages. JISC extended our project end-date, to better mesh with an anticipated revamp of associated services within the host unit. That revamp 's been

JPEGs are created by taking snapshots of the records.



## AstroDAbis

Creation Date: 2013-08-22

### Original blog URL

<http://astrodabis.jiscinvolve.org/wp>

<b>AstroDAbis</b>	
<small>Stand-off annotation for astronomical catalogues</small>	
<b>PAGES</b>	<b>AstroDAbis's beta service</b>
<small>About Resources and outputs</small>	<small>March 22nd, 2012</small>
<b>ARCHIVES</b>	<small>The AstroDAbis project's annotation service is now up and running at <a href="http://astrodabis.roe.ac.uk">astrodabis.roe.ac.uk</a>, for your delight and</small>
<small>March 2012</small>	

## Implementation details

- LaTeX TeX Live distribution is used for creating PDF files. “install-texlive” target is added to **Makefile.am** to install required packages.
- PhantomJS is used for taking snapshot of the pages to create JPEG files. “install-phantomjs-64bits” and “install-phantomjs-32bits” targets are added to **Makefile.am** to install required packages.
- **Default\_HTML\_actions.bft** is modified to add PDF and JPEG as an exporting option.
- Four new output formats are introduced:
  - **JPEG.bfo** : Output format to construct the page for taking snapshot.
  - **JPEGB.bfo** : Like **JPEG.bfo** but less detailed.
  - **PDF.bfo** : Output format to construct LaTeX template.
  - **PDFB.bfo** : Like **PDF.bfo** but less detailed.
- The following format templates are introduced to create LaTeX documents and the page to take snapshot for exporting as JPEG:
  - **JPEG.bft**
  - **BlogJPEG.tpl**
  - **CommentJPEG.tpl**
  - **Comment\_JPEG\_brief.tpl**
  - **PostJPEG.tpl**
  - **Post\_JPEG\_brief.tpl**
  - **LaTeX.bft**
  - **BlogLaTeX.tpl**
  - **CommentLaTeX.tpl**
  - **Comment\_LaTeX\_brief.tpl**
  - **PostLaTeX.tpl**
  - **Post\_LaTeX\_brief.tpl**
- The following BibFormat elements are introduced for creating LaTeX documents:
  - **bfe\_latex\_abstract.py**
  - **bfe\_latex\_authors.py**
  - **bfe\_latex\_begin.py**
  - **bfe\_latex\_blog\_posts.py**
  - **bfe\_latex\_blog\_url\_link.py**
  - **bfe\_latex\_comment\_header.py**
  - **bfe\_latex\_default.py**
  - **bfe\_latex\_end.py**
  - **bfe\_latex\_post\_comments.py**
  - **bfe\_latex\_record\_body.py**

- `bfe_latex_record_dates.py`
  - `bfe_latex_snapshot.py`
  - `bfe_latex_tags.py`
  - `bfe_latex_title.py`
- The following BibFormat elements are modified to create the page and take snapshot for exporting as JPEG:
    - `bfe_blog_posts.py`
    - `bfe_post_comments.py`
    - `bfe_record_dates.py`
    - `bfe_snapshot.py`
  - The following templates are created to export as JPEG and PDF.
    - `footer_jpeg.html` : Footer used for exporting as JPEG.
    - `header_jpeg.html` : Header used for exporting as JPEG.
    - `page_jpeg.html` : The base page to inherit format templates for exporting as JPEG.
    - `page_latex.tpl` : The base page to inherit format templates for exporting as PDF.
    - `record_jpeg.html` : Default template to export as JPEG.
  - To get the export requests for PDF and JPEG, `index` endpoint of `websearch_blueprint.py` and `print_records` function in `search_engine.py` is modified.
  - `create_pdf` and `create_jpeg` functions are introduced in `bibformat.py` to create PDF and JPEG files, respectively.
  - `summary` endpoint is introduced in `record_blueprint.py` to construct the page for taking snapshot to export as JPEG.
  - `jpeg_render_script.js` is introduced to use on taking snapshot of the page by PhantomJS.
  - `latexutils.py` and `latexutils_image.py` is created to keep common functions to creating LaTeX document. The following functions are in `latexutils.py` :
    - `begin_document` : Initialization codes for a LaTeX document.
    - `end_document` : Finalization codes for a LaTeX document.
    - `remove_escape_chars` : Returns LaTeX representation of special LaTeX characters.
    - `format_raw_text` : Formats given text as LaTeX.
    - `format_latex_field` : Formats given name and value to display as BibTeX field.
    - `html_to_latex` : Converts HTML to LaTeX document.

The following functions are in `latexutils_image.py` :

- **get\_image\_path** : Returns the path of given image. If the image does not exist in file system, downloads it to an available location.
- **get\_unique\_file\_path** : Returns a unique path to save the image temporarily.
- To convert HTML to LaTeX, **bibformat\_pdf\_with\_latex\_template.py** module is created. It has the following classes:
  - **PdfWithLatexTemplateHtmlParser** class is inherited from **HTMLParser** class.
  - **LatexConverter** : Converts HTML to LaTeX. Possible LaTeX commands are mapped from dictionaries in **bibformat\_pdf\_with\_latex\_config.py** module. Only a few HTML tags are needed extra effort. The methods defined to handle them are called from **PdfWithLatexTemplateHtmlParser** class. It has the following methods:
    - \* **get\_latex\_text** : Returns concatenation of definitions and latex codes.
    - \* **\_handle\_latex\_commands** : Adds given data to one of the buffers holding LaTeX template according to state of HTML tags.
    - \* **handle\_raw\_data** : Main controller for inserting HTML elements' content.
    - \* **\_insert\_raw\_data** : Handles LaTeX special characters, maps special HTML characters (e.g. &hellip) with their LaTeX equivalences and adds the content to latex code buffer.
    - \* **trivial\_tag\_start** : Handles start tags.
    - \* **tag\_end** : Pops end values until popped element's type is given tag. Until an HTML tag is found, CSS related values are added to buffer. For table related elements extra effort is needed.
    - \* **br\_start** : Applies new line only after text.
    - \* **font\_start** : Handles attributes of font tag.
    - \* **anchor\_start** : Handles attributes of anchor tag.
    - \* **img\_start** : Handles img tag based on parameters in config module. If **CFG\_BIBFORMAT\_LATEX\_USE\_LOCAL\_IMAGES** is not set, then the image is downloaded from src attribute value. If the image extension is not supported by LaTeX, converts image into specified format.
    - \* **table\_start** : Handles start tag of table.
    - \* **table\_end** : Handles end tag of table.
    - \* **table\_row\_start** : Handles start tag of tr.
    - \* **table\_row\_end** : Handles end tag of tr.
    - \* **table\_cell\_start** : Handles start tag of td or th.
    - \* **flush\_end\_stack** : If there is not any problem with the HTML content, all HTML tags are successfully matched. If there are any element for style rules, pops them and adds 'end' values to LaTeX text.

- \* **add\_style** : Adds style related LaTeX codes based on current tag's CSS rules. Most of the CSS declarations are mapped from dictionary. However, some needs extra effort (e.g. font-size, color, border, etc.)
- \* **\_\_search\_style\_in\_external\_css** : In all possible selectors' CSS rules, searches given property.
- \* **extract\_style** : Finds all possible CSS declarations.
- \* **find\_style\_selectors** : Checks current HTML element's parents and their ids, class names to construct CSS selectors possibly containing more than one identifier (e.g. ".class\_name >#id", "div div img", "div.class\_name >p", etc.)
- **CssParser** : Parses the CSS rules and converts into dictionary. It has the following methods:
  - \* **parse** : Finds each CSS declaration and selectors of CSS rules.
  - \* **parse\_inline\_style** : Rather than whole css file data, runs on only inline CSS.
  - \* **\_\_extract\_css\_declarations** : Converts style rule declarations in str form to dictionary.
  - \* **\_\_read\_css\_files** : Reads CSS file. This method is called when the data is supplied as file names.
- Configurations for converting HTML to LaTeX take place in **bibformat\_pdf\_with\_latex\_template\_config.py** . It has the following configurations:
  - **CFG\_BIBFORMAT\_EXPORT\_DIR** : The directory to keep files of exported record.
  - **CFG\_BIBFORMAT\_LATEX\_TEMP\_DIR** : The directory to keep temporary files when creation PDF from LaTeX.
  - **CFG\_BIBFORMAT\_PATH\_PDF\_CONVERTER** : The path of *xelatex* tool.
  - **CFG\_BIBFORMAT\_LATEX\_KEEP\_FILE\_EXTENSIONS** : The file extensions which will be kept in **CFG\_BIBFORMAT\_EXPORT\_DIR** after *pdflatex* ends its job.
  - **CFG\_BIBFORMAT\_CSS\_FILES** : The paths of the CSS files used when converting HTML to LaTeX template.
  - **CFG\_BIBFORMAT\_IMG\_SRC\_URL** : Regular expression to check *src* attribute of *img* whether it is a local path or a URL.
  - **CFG\_BIBFORMAT\_LATEX\_HEADER** : The first line of the LaTeX document.
  - **CFG\_BIBFORMAT\_LATEX\_PAGE\_SETTINGS** : Options for page layout.
  - **CFG\_BIBFORMAT\_LATEX\_BEGIN\_DOC** : The *begin* tag of the LaTeX document.
  - **CFG\_BIBFORMAT\_LATEX\_END\_DOC** : The *end* tag of the LaTeX document.

- **CFG\_BIBFORMAT\_LATEX\_PACKAGES** : The packages that are used in LaTeX document.
- **CFG\_BIBFORMAT\_LATEX\_SPECIAL\_CHARS\_REGEX** : Regular expression to identify special characters in LaTeX.
- **CFG\_BIBFORMAT\_LATEX\_SPECIAL\_CHARS\_LIST** : The list of special characters in LaTeX.
- **CFG\_BIBFORMAT\_LATEX\_MATH\_FORMULAS** : Regular expression to find patterns of math formulas which are in already LaTeX format.
- **CFG\_BIBFORMAT\_LATEX\_ONLY\_TEXT** : Regular expression to split content into pieces according to start and end of math formulas.
- **CFG\_BIBFORMAT\_LATEX\_LONG\_WORDS** : Regular expression to find words containing symbols more than provided number.
- **CFG\_BIBFORMAT\_MATHJAX\_ENABLED** : Set to 1 to seek LaTeX formulas in text. If it is set to 0, then **CFG\_BIBFORMAT\_MATHJAX\_DELIMITERS** is ignored.
- **CFG\_BIBFORMAT\_MATHJAX\_DELIMITERS** : Delimiter characters for formulas.
- **CFG\_BIBFORMAT\_CSS\_PARSER\_REGULAR\_EXPRESSIONS** : Regular expressions used for parsing CSS.
- **CFG\_BIBFORMAT\_CSS\_SELECTOR\_COMPONENTS** : Regular expression to identify the components of a CSS selector.
- **CFG\_BIBFORMAT\_LATEX\_CSS\_SELECTORS** : Complex selector types for which CSS rules will be applied.
- **CFG\_BIBFORMAT\_LATEX\_STYLE\_FIRST\_TAGS** : The list of the tags to apply CSS rules before handling.
- **CFG\_BIBFORMAT\_LATEX\_NON\_END\_TAGS** : The tags that may not have '/' character indicating end of tag.
- **CFG\_BIBFORMAT\_LATEX\_EXCEPTIONAL\_COMMANDS** : List of commands after which an environment can not be initialized.
- **CFG\_BIBFORMAT\_LATEX\_DEFAULT\_IMG\_FORMAT** : Default image format for LaTeX.
- **CFG\_BIBFORMAT\_LATEX\_SUPPORTED\_IMG\_EXTS** : The list of supported image types.
- **CFG\_BIBFORMAT\_LATEX\_PX\_CM\_SCALE** : Cm value that corresponds to 1 pixel.
- **CFG\_BIBFORMAT\_LATEX\_A4\_SCALE** : Amount of scale for A4 paper dimensions.
- **CFG\_BIBFORMAT\_LATEX\_DEFAULT\_IMG\_SCALE** : The rate to scale the image.

- **CFG\_BIBFORMAT\_LATEX\_DEFAULT\_FONT\_SIZE** : Default font size for LaTeX document.
- **CFG\_BIBFORMAT\_LATEX\_KB\_FILE** : The path of the latex-to-unicode.kb file.
- **CFG\_BIBFORMAT\_LATEX\_SPECIAL\_CHARS\_EQUIVALLANCES** : Representations of special LaTeX characters.
- **CFG\_BIBFORMAT\_LATEX\_COMMAND\_CONSTRAINTS** : Constraints for LaTeX tags.
- **CFG\_BIBFORMAT\_LATEX\_ALIGNMENTS** : LaTeX representations of HTML alignment options.
- **CFG\_BIBFORMAT\_LATEX\_COMMANDS** : LaTeX commands used for rather than tag and CSS mappings.
- **CFG\_BIBFORMAT\_LATEX\_REPRESENTATION\_OF\_HTML\_TAGS** : HTML tags and their LaTeX representations.
- **CFG\_BIBFORMAT\_LATEX\_REPRESENTATION\_OF\_CSS\_RULES** : CSS declarations and their LaTeX representations for possible values.
- **CFG\_BIBFORMAT\_HTML\_SPECIAL\_CHARS\_BY\_NUM** : Unicode numbers of special HTML chars and their LaTeX representations.
- **CFG\_BIBFORMAT\_HTML\_SPECIAL\_CHARS\_BY\_NAME** : Special HTML chars and their LaTeX representations.

<b>Implemented By</b>	Senan Postaci(SRDC)
-----------------------	---------------------

Table 4.13: Implementation Description: RF62

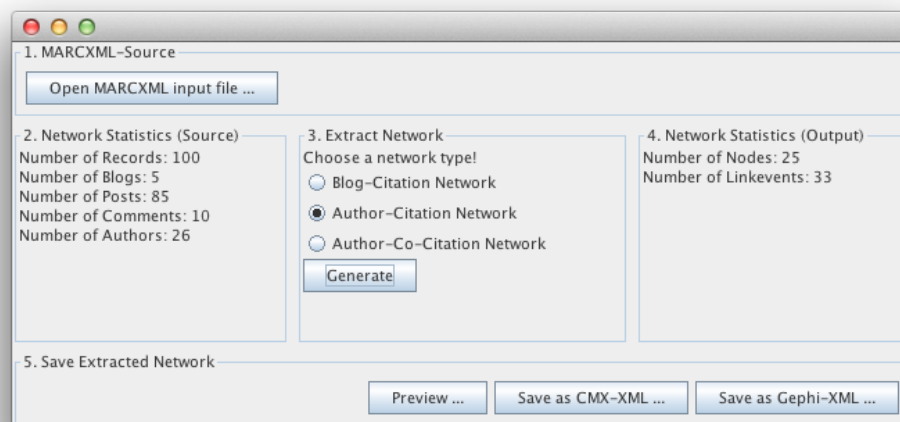
<b>Feature ID</b>	RF65 (Repository Feature 65)
<b>Name</b>	The archive analyzes blog links and stores the connections between them separately
<b>Effort Spent</b>	4 weeks
<b>Modules Affected / Created</b>	-
<b>Description of the new feature</b>	
<p>A network consists of nodes and links between the nodes. Three kinds of networks are extracted from the archived blogosphere:</p>	



- Blog-Citation network
  - Nodes: All Blog objects in the archived blogosphere.
  - Links: A link is created when a hyperlink in a post references another post or a blog. The link goes from the parent blog of the sending post to the parent blog of the receiving post or the receiving blog.
- Author-Citation network
  - Nodes: All authors in the archived blogosphere.
  - Links: A link is created when a hyperlink in a post or in a comment references another post or comment. Sending and receiving posts or comments must have an author.
- Author-Co-Citation network
  - Nodes: All author objects in the archived blogosphere.
  - Links: A link is created if two or more posts or comments reference the same address through a hyperlink.

The Blog-Citation network and the Author-Citation network consider only links that direct to an object that is archived in the repository. The Author-Co-Citation network considers all kind of links (external and internal).

The implementation of the feature differs from the feature design: Links are not differentiated between Citations, BlogRoll, and Pingback/Trackback because these information can not be identified by the spider.



RF65 - GUI of the feature. The main steps in the network generating process are (a) to choose the input, (b) to choose the network type, and (c) to choose the output format and location

### Implementation details

The feature is implemented as java application together with the repository feature RF72 which visualizes the extracted network. It has been also ported to Python and this is the version that has been integrated into the repository.

Nodes are represented as objects that have an ID and a maximum of five details. Links are an aggregation of one or more Linkevents. A linkevent has one sender, one or more recipients, a timestamp, and a maximum of five details. The distinction of Linkevents with timestamps allows representing and analysing the network evolution over time.

The application accepts as input a collection of blog, post, and comment objects in MARXML from the repository. The objects in the collection are analyzed and the network extracted by the Java class *tub.BFMarcXMLReader* and stored in the Java objects

- *tub.dataElements.Network*
- *tub.dataElements.Node*
- *tub.dataElements.Linkevent*

Currently, the two output formats CMX-XML and GEXF are provided.

- With CMX-XML, the extracted network can be further analysed and explored with the Commetrix<sup>a</sup> software. The Java class *tub.CMXMLWriter* transforms the extracted network into CMX-XML. The CMX-XML preserves the structure of link events, and, therefore, Commetrix facilitates dynamic and static analysis.
- GEXF can be analysed with Gephi<sup>b</sup>, an open-source tool to visualize and analyse networks. The Java class *tub.GEXFWriter* transforms the extracted network into GEXF. Thereby, the linkevents has to be aggregated to edges due to the limitations of the GEXF format. Therefore, the Gephi tool facilitates only static analysis.

For each of these two output formats a BibFormat template has been created: *format\_records\_cmx.tpl* and *format\_records\_gexf.tpl*

The Java class *tub.BFNetworkGenerate* contains the GUI for the Network Generator application.

<sup>a</sup><http://www.commetrix.de/>

<sup>b</sup><https://gephi.org/>

<b>Implemented By</b>	TUB
-----------------------	-----

Table 4.14: Implementation Description: RF65

<b>Feature ID</b>	RF73 (Repository Feature 73)
<b>Name</b>	The archive recommends blogs to users based on the ratings and preferences
<b>Effort Spent</b>	1 Week
<b>Modules Affected / Created</b>	BibRank, WebSearch

### Description of the new feature

This feature introduces a new ranking method to rank the records by their weighted averages.

A portalbox which shows the Top Rated Records has been added into main page.

#### Top Rated Records

**3.3459 Question 2 Exam 2**  
**3.3321 Exam 1 Question 1**  
**3.3309 BULLYING-RESTORATIVE JUSTICE**  
**3.3294 Catering for a difference websites**  
**3.3289 Basic Assumption (Dreikurs)**  
**3.3248 EDUCATION FOR THE 21ST CENTURY-WORKING WITH GROUPS**  
**3.3233 A poke with a sharp stick | by david carter-tod**  
**3.3126 A Difference**  
**3.3081 Exactly what is Emotional Intelligence?**  
**3.3074 Glasser and Classroom management.**

RF73 - Portalbox displaying top rated records

A portalbox which shows last added records has been added into main page.

#### Last Added Records

**2013-05-24 17:35 The Hyper-Social Organization**  
**2013-05-24 17:34 How to Position Enterprise 2.0**  
**2013-05-24 17:34 Potential Benefits of Enterprise 2.0**  
**2013-05-24 17:34 What will power next-generation businesses?**  
**2013-05-24 17:34 Three Secret Weapons Of Innovation**  
**2013-05-24 17:34 Telstra's 3Rs of Social Media Engagement**  
**2013-05-24 17:34 Tactics for improving customer engagement**  
**2013-05-24 17:34 Gary Hayes's social media counter**  
**2013-05-24 17:34 Google Wave Versus the Rest, Feature by Feature**  
**2013-05-24 17:34 A CMO's Guide To The Social Media Landscape**

RF73 - Portalbox displaying recently added records

## Implementation details

- Portal boxes have become associated with ranking methods.
  - `bibrank_portalbox` table, which keeps which ranking method is related with which portal box, has been inserted into database.
  - `update_bibrank_portalbox` and `drop_bibrank_portalbox` functions have been added into `bibrank_record_sorter.py` .
    - \* `update_bibrank_portalbox` function updates the portal boxes when bibrank is run.
    - \* `drop_bibrank_portalbox` function removes the entries related to given bibrank method when either the ranking method is deleted or the variable keeping the number of records shown in portalbox is set to `0` .
  - For portalboxes `bibrank_portalbox.html` template is added.
- Ranking with “Weighted Average” has been introduced.
  - To rank the records by their weighted average, `bibrank_weighted_average_indexer.py` module has been created. This module contains the function `weighted_average_to_index` which calculates the weighted average with “Bayesian estimate” which is the following formula:
 
$$\frac{N}{N + m} * A + \frac{m}{N + m} * G$$
 where
    - \* **N**: the number of reviews of corresponding record
    - \* **m**: minimum number of reviews required to calculate the rank of the record
    - \* **A**: Average score of corresponding record
    - \* **G**: Average score of all of the records
  - `bibrank_weighted_average_template.cfg` containing the parameters for the ranking method has been created. These parameters are:
    - \* `show_relevance` : `1` to show the score on search page, `0` otherwise.
    - \* `minimum_review_number` : minimum number of reviews required to be ranked
    - \* `display_on_portalbox_count` : the number of the records will be displayed on the portalbox. If it is `0` , portalbox disappears and entries related to that ranking method is removed from database.
- Archived content indexer has been added as a ranking method to create “Recently Added Records” portalbox.

- **bibrank\_archived\_content\_indexer.py** has been introduced to rank the records in a time interval.
- **template\_recently\_archived\_content.cfg** containing the parameters for the ranking method. These parameters are:
  - \* **latest\_records\_number** : the number of the lastly added records, if **0** , ranks all of them.
  - \* **date\_type** : **creation** for ranking by creation date, **modification** for ranking by modification date.
  - \* **start\_date** : the beginning of the time interval.
  - \* **end\_date** : the end of the time interval.
  - \* **interval** : the sql like time interval. (i.e. **3 HOUR** , **1 DAY** )
  - \* **display\_on\_portalbox\_count** : the number of the records will be displayed on the portalbox. If it is **0** , portalbox disappears and entries related to that ranking method is removed from database.
- Some modifications have been occurred in **bibrank.py** and **bibrank\_tag\_based\_indexer.py** to accommodate new ranking method.

<b>Implemented By</b>	SRDC
-----------------------	------

Table 4.15: Implementation Description: RF73

## Chapter 5

# Future Work

The platform designed and developed by the WP4 team is complete and presents a set of features that satisfy the requirements of the DoW and the requirements of the stakeholders interviewed in D4.1[4]. However, some features were not included in the Repository given that the complexity of the topic is falling out of scope of blog archiving platform. These features could be integrated with the blog archive in later stages of development using already existing solutions or developing blog-specific solutions to live within the BlogForever platform code. This possibility was already pointed out in Section 3.3 of D4.7[1]. These features are:

- *RF42 - The archive extracts bibliographic metadata from content embedded in blogs:* The BlogForever archive would extract bibliographic metadata (e.g. Title, Author) from PDF documents, LaTeX files and image files that are attached to blogs, or embedded in the post of a blog. The extracted metadata would be stored in the database and used to populate an index. Users would be able to search and browse the metadata for the attachments, using a special web interface.
- *RF68 - The archive provides information diffusion analysis mechanisms:* Memetracking and trend detection. The archive would provide the user with a set of tools that allow detecting the provenance of various topics, phrases and memes. In addition to their provenance, the archive would provide the full history of a given meme, allowing for the user to adjust the time of content and study the spread and the diffusion of information in the archive. The archive would detect influential memes and would process them according to time.
- *RF75 - The archive can do sentiment analysis on the content:* To give the users an idea about the blog post's attitude, sentiments of the post should be analyzed. Sentiment analysis would be run by the administrator. The user would be able to see the sentiment scores of the contents. The users would be able to add a search criteria that ranks the results based on their sentiment scores.

On top of these features, there is a number of items where an extra development effort would push the platform one step higher in terms of data quality and user interface completeness.

- **Parallel ingestion:** In order to speed-up the ingestion throughput, the ingestion plugins could be extended to support the upload of multiple records in the same task. This would increase the scalability in terms of the number of daily new posts that the Repository is able to ingest without the need of a multi-server infrastructure.
- **Repository sharing:** In order to scale repository to cover the use cases of tens of millions of records, a repository sharing technique can be considered. This feature has been developed recently by the core Invenio team [13]. This would provide additional scalability complement to the above mentioned parallel ingestion. (Horizontal vs vertical scaling.)
- **Metadata curation daemon:** When Post records are inserted into the Repository, there is a search to retrieve the parent Blog record identifier. This way, the post-blog relationship is recorded in the metadata. This parent blog lookup sometimes fails leaving an orphan post record. Developing a daemon that goes over orphan records and fixes their metadata would not just contribute to better metadata but also would permit parallel ingestion of records without the need of checking if the parent record has already been inserted into the repository.
- **Pages:** Enhance the Spider component for better detection of Comments and specially Pages (the pages of a blog that are not entries, like the “about” page).
- **Topic detection:** We have implemented a prototype for topic detection. The prototype is based on the state-of-the-art approach of *Latent Dirichlet Allocation* (LDA). LDA typically takes as input a corpus of documents and a fixed *number* of topics to be induced. The output is the topics, which is provided as a collection of terms-words for each topic. Furthermore, during the training process, a model is generated which can then be fed with a new document in order to classify the latter based on the topics generated. In general, in order to produce a good model and have satisfying results, a fairly big collection of documents (in the scale of tens of thousands) must be used and a good estimation on the expected number of topics must be made.

Our prototype is developed using Python and is integrated to consume Invenio API calls. More specifically, the core library is gensim<sup>1</sup> along with some required modules (logging, numpy, scipy). In addition to the above, a language detection module (langid) was chosen, in order to generate distinct models (and topics) for each language.

In our experiments, gensim library showed promising results. This is inline to the findings of project ARCOMEM (deliverable 3.2<sup>2</sup>), where the authors claim that LDA should yield meaningful topics. In their approach, they report poor results, mainly due to the content of HTML documents not being cleaned up and not having boilerplate text removed. In our experiments, we have

---

<sup>1</sup><http://radimrehurek.com/gensim/>

<sup>2</sup><http://www.arcomem.eu/wp-content/uploads/2012/05/D3.2.pdf>

managed to extract<sup>3</sup> the main content of blog posts and have applied LDA on them. Finally, an issue worth discussing, which was not fully addressed in our implementation, is the presentation and visualisation of the topics themselves. As provided by the gensim library, LDA is actually describing the topics with a long list of terms/words; in our approach we would have preferred to describe the topics with fewer words. Therefore, further tuning and customisation of the implemented solution is proposed. Furthermore, we expect that when deployed on a real-world dataset, some experimentation with the number of topics should take place.

- **Statistics:** The repository databases implicitly keep many statistics that could be extracted and used to enrich the statistics tab of the detailed view of a record with interesting information.
- **Highlight share:** The highlighting feature is very nice, allowing a user to highlight part of the text content using different colors and add notes to them. According to the feedback received from the testers a good add-on would be the possibility of sharing these highlights and annotations with other users of the platform.
- **Invenio code updates:** Since the Repository is based on Invenio it inherits all its good features, but it might also get some small bugs. The Invenio team works hard to minimize them, so rebasing the repository code against the latest Invenio code is highly recommended. This way, the BlogForever Repository will benefit from the latest Invenio features and the possible bugs would be fixed.

---

<sup>3</sup>BlogForever has implemented a “main content” extraction technique - See D2.6 for more details.



## Chapter 6

# Conclusions

The aim of this report was to present the integration of the Weblog Digital Repository Component and the Weblog Spider Component as the final outcome of the BlogForever project — and WP4 in particular. It presents the final BlogForever Software Infrastructure that culminates the work of gathering the requirements and specifications described in D4.1[4]; design and implementation of the Spider component described in D4.2[9], D4.3[10] and D4.6[2]; and design and implementation of the Repository component described in D4.4[5], D4.5[8] and D4.7[1].

The ability of both components to communicate with other pieces of software has been exposed in Chapter 2. While the Spider Component offers an API to manage the list of blog URLs to monitor and to download the crawled data, the Repository Component supports a plugin system that lets potential developers to write the bridge they need to communicate with any crawler or blog data provider. This shows that although both components work nicely together they are not depending on each other.

Chapter 3 presents the specific way in which the spider-repository communication has been implemented using the mechanisms introduced in Chapter 2. The chapter reflects the use of the Spider API by the repository to submit and monitor new blog URLs and the fetcher used to retrieve data, as well as the tasks performed in the pre-ingestion and post-ingestion plugins.

The latest development is presented in Chapter 4. While the Spider component has not suffered any notable modifications, the Repository new features and updates have been reported, including their implementation description. The development has worked in close collaboration with the testing performed by WP5, gathering valuable feedback and applying it to the code in order to improve both stability and usability.

Finally, a series of suggestions on how to further improve the platform has been pointed out in Chapter 5.

# References

- [1] J. García Llopis, R. Jiménez Encinar, S. Postacı, A. Çınar, H.Kalb, and T. Šimko, “*D4.7: Final Weblog Digital Repository Component*,” work package, European Organization for Nuclear Research (CERN), May 2013. Work Package Four Deliverables.
- [2] M. Rynning, “*D4.6: Final Weblog Spider Component*,” work package, CyberWatcher, Apr. 2013. Work Package Four Deliverables.
- [3] “Blogforever,” tech. rep., 2011.
- [4] H. Kalb, N. Kasioumis, J. García Llopis, S. Postacı, and S. Arango-Docio, “*D4.1: User Requirements and Platform Specifications Report*,” work package, B. Consortium (Ed.): Technische Universität Berlin, Dec. 2011. Work Package Four Deliverables.
- [5] J. García Llopis, R. Jiménez Encinar, *et al.*, “*D4.4: Digital Repository Component Design*,” work package, European Organization for Nuclear Research (CERN), Nov. 2012. Work Package Four Deliverables.
- [6] K. Stepanyan, M. Joy, A. Cristea, Y. Kim, E. Pinsent, and S. Kopidaki, “*D2.2: Weblog Data Model*,” work package, B. Consortium (Ed.): University of Warwick (UW), Oct. 2011. Work Package Two Deliverables.
- [7] Y. Kim, S. Ross, K. Stepanyan, E. Pinsent, P. Sleeman, S. Arango-Docio, H. Kalb, *et al.*, “*D3.1 Preservation Strategy Report*,” work package, Y. Kim & S. Ross (Eds.): University of Glasgow, Sept. 2012. Work Package Three Deliverables.
- [8] S. Postacı, A. Çınar, G. B. Laleci, J. García Llopis, R. Jiménez Encinar, V. Banos, and A. P. and, “*D4.5: Initial Weblog Digital Repository Prototype*,” work package, SRDC Yazilim Arastirma ve Gelistirme ve Danismanlik Ticaret Limited Sirketi, Jan. 2013. Work Package Four Deliverables.
- [9] H. Kalb, P. Lazaridou, and M. Trier, “*D4.2: Weblog spider component design*,” work package, B. Consortium (Ed.): Technische Universität Berlin (TUB), June 2012. Work Package Four Deliverables.
- [10] M. Rynning, “*D4.3: Initial Weblog Spider Prototype*,” work package, CyberWatcher, Sept. 2012. Work Package Four Deliverables.
- [11] S. Arango-Docio, P. Sleeman, E. Pinsent, G. Gkotsis, T. Farrell, S. Kopidaki, and M. Rynning, “*D5.1: Design and Specification of Case Studies*,” work package, University of London, June 2012. Work Package Five Deliverables.

- [12] T. all BlogForever project partners, “*D6.5: Business and exploitation plan,*” work package, Tero Ltd, Aug. 2013. Work Package Six Deliverables.
- [13] V. C. Venkatraman, “*Enhancing Scalability of Invenio - Digital Library Software,*” tech. rep., École Polytechnique Fédérale de Lausanne (EPFL), May 2013.

## Appendix A

# Final Repository Implementation Descriptions

One of the objectives of this deliverable is to present the implementation activities of the whole suite of repository features defined in D4.4[5] In order to include the description of the features along with its implementation details, the same template that was created in D4.5[8] will be used.

<b>Feature ID</b>	RFID (Repository Feature ID)
<b>Name</b>	One sentence clear enough to make someone who has already read the specification remember the description
<b>Effort Spent</b>	Actual implementation time (Possible values: Days/Weeks/Months)
<b>Modules Affected / Created</b>	Name of the Invenio modules either modified or introduced
<p><b>Description of the new feature</b></p> <p>High level description of the feature. How Invenio is extended or what kind of functionality is introduced, is described here. A general screenshot indicating the general execution of the new feature may be included here.</p> <p><b>Implementation details</b></p> <p>Technical details of the implementation activities are described here. All the files and modules that are exposed to modifications (i.e adding/ altering classes/methods, introducing new fields into configuration files, new user interfaces, etc) and how they are modified are explained in detail. Screenshots of new functionalities are provided here.</p>	
<b>Implemented By</b>	Person or partner who implemented the feature

Table A.1: Implementation Description: RFID

Section A.1 lists the features that Invenio already provides and fulfills our needs, Section A.2 presents the implementation descriptions of the whole list of repository features that were defined in D4.4[5], and Section A.3 lists those repository features that was decided not to implement, as well as the reasons why this decision was taken.

## A.1 Features already in Invenio

As mentioned earlier, Invenio is a comprehensive software for digital library management. Therefore, it already supports 34 of the repository features. However, some of these features needed to be configured in order to meet the requirements of the final BlogForever platform. These features are listed as follows:

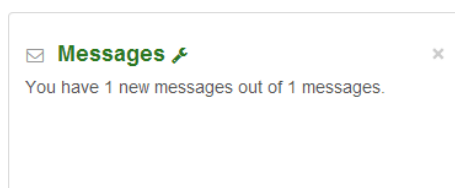
- RF5 - The web interface provides harmonized access and ensures compatibility with major browsers
- RF7 - Export data using the OAI-PMH protocol
- RF8 - Export data using Dublin Core Schema
- RF10 - Archive user passwords are stored encrypted in the database
- RF11 - The archive web interface is available in many different languages
- RF13 - UTF is used as the default character encoding in the archive
- RF15 - Option to disseminate archive content in major social web platforms
- RF16 - The archive offers an RSS channel of its latest updates and/or users can receive notification when new content of their interest is added to the archive
- RF18 - The archive detects duplicated content and keeps only one copy
- RF19 - The archive can be indexed by external search engines
- RF20 - The archive's statistics are exported as CSV
- RF21 - The archive offers the option to login using SSO / LDAP
- RF27 - The archive displays a unique URL (DOI) for each record
- RF29 - The archive alerts the user when there are software updates
- RF33 - The archive can display only the very core information for each record
- RF37 - The archive restricts the access to its content to specific IP ranges
- RF38 - Users can communicate within the archive sharing and exchanging resources
- RF39 - Free open-source archive software
- RF43 - For each record the archive stores the search keywords used to find them
- RF44 - The archive enables pingback/trackback services
- RF51 - The archive is able to search within external sources external collections, hosted collections

- RF55 - The archive provides advanced APIs for use by developers to interact with the archive's content
- RF60 - The archive can export all its content, database entries and file system for migration
- RF69 - The archive facilitates searching by providing fuzzy indexing and stemming
- RF74 - The archive enables/disables certain functionalities based on the content rights
- RF77 - The archive provides a mobile version
- RF79 - The archive can handle a very large number of content and users
- RF80 - The archive provides mechanisms to control data redundancy
- RF81 - The archive is built based on a modular service-oriented architecture
- RF82 - The archive can be deployed using a range of different database server technologies
- RF83 - The archive provides multiple different views of the archive for each user
- RF84 - The archive offers a complete range of search options to the user
- RF85 - The archive provides support for OpenURL
- RF86 - The archive offers functions to edit metadata

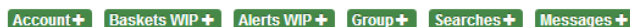
## A.2 List of implementation descriptions

In this Section the final set of implementation descriptions of the final repository features is presented.

<b>Feature ID</b>	RF1 (Repository Feature 1)
<b>Name</b>	Customizable user dashboard
<b>Effort Spent</b>	Days
<b>Modules Affected / Created</b>	WebSession
<b>Description of the new feature</b>	
<p>The layout of the dashboard is re-organized. Floating layout has become 3-column layout so that each column become independent from each other.</p> <p>Each box in user dashboard has become collapsible. They can be collapsed by clicking 'x' symbol on top right corner of the box.</p>	



The list of the collapsed boxes takes place on top of the dashboard. A collapsed box can be retrieved back by clicking on its name.



### Implementation details

- To change the layout of the user dashboard `webaccount_display.html` is modified and `webaccount_widget.html` is created.
- The `index` endpoint of `webaccount_blueprint.py` is modified to display the dashboard boxes in proper column.
- To expand the collapsed boxes in user dashboard, `loadwidget` endpoint is introduced in `webaccount_blueprint.py`.

<b>Implemented By</b>	Alper Çınar (SRDC)
-----------------------	--------------------

Table A.2: Implementation Description: RF1

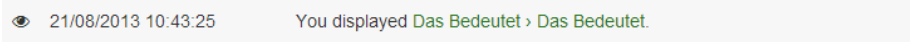
<b>Feature ID</b>	RF2 (Repository Feature 2)
<b>Name</b>	“Your History” box as part of the user dashboard
<b>Effort Spent</b>	Weeks
<b>Modules Affected / Created</b>	WebSession

### Description of the new feature

“Your Activities” page is re-constructed while porting this feature to next. This page includes the user’s history in reverse chronological order with 15 main categories:

- Adding into basket
- Basket creation/subscription
- Notes on baskets
- Votes

- Page views
- Searches
- Subscriptions
- Downloads
- Messages
- Alerts
- Discussions
- Reviews
- Reports
- Groups
- Payments

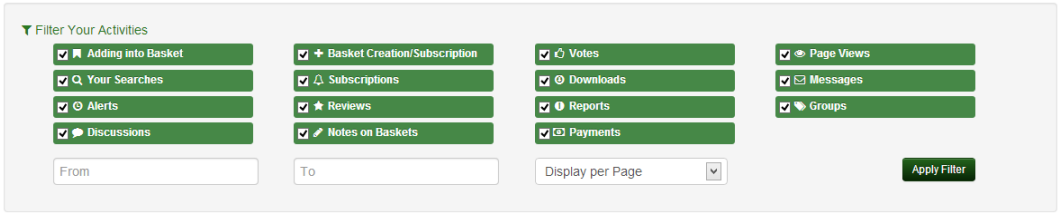


By default, the most recent 10 activities are displayed on “Your Activities” page. User may load older activities through “Load older activities” link on the bottom of the page.



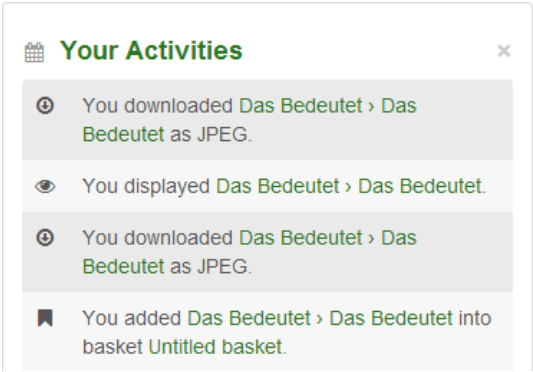
RF2 - The link to display older activities.

A filter is created to filter the categories that are displayed on “Your Activities” page.



RF2 - Filter panel

The new box for “Your Activities” is created in the user dashboard. It consists of last 10 activities of the user.





## Implementation details

- **HistoryElement** class is introduced to represent each history item in the “Your Activities” page. It has only one constructor and the following instance variables:
  - **date** : Date of the history element.
  - **entry** : Database entry corresponds to that history element.
  - **id** : ID of the entry.
  - **history\_type** : Type of the history.
  - **icon** : Icon that is displayed near the activity message on the web interface.
  - **msg** : Message that is displayed on the web interface.
- **HistoryCollector** abstract class is introduced to create History Collectors for each category. It has the following instance variables and methods:
  - **icon** : Icon for collected **HistoryElement** s.
  - **label** : Label to filter this collector.
  - **check** : Returns the most recently fetched **HistoryElement** s. (This method should not be overridden in its child classes)
  - **use** : Fills the required attributes of the most recently fetched **HistoryElement** and returns it. (This method should not be overridden in its child classes)
  - **get\_user\_history** : Fetches and returns entries from database.
  - **get\_id** (static): Returns the ID of the given entry.
  - **get\_date** (static): Returns the date of the given entry.
  - **get\_message** (static): Returns the message displayed on web interface.
- 15 new classes are inherited from **HistoryCollector** class to collect user history. These are:
  - **PaymentHistory**
  - **AlertHistory**
  - **BasketCreationHistory**
  - **BasketNoteHistory**
  - **BasketAddHistory**
  - **DiscussionHistory**
  - **ReportHistory**
  - **ReviewHistory**
  - **SubscriptionHistory**
  - **VoteHistory**

- [MessageHistory](#)
- [DownloadHistory](#)
- [SearchHistory](#)
- [RecordViewHistory](#)
- [GroupHistory](#)
- [HistoryManager](#) class introduced to collect the user history from registered [HistoryCollector](#) s. It has the following methods:
  - [get\\_user\\_history](#) : Fetches and returns the list of the user history.
  - [has\\_mote\\_history](#) : Returns if the user has more [LHistoryElements](#) to fetch.
  - [get\\_user\\_history\\_json](#) : Returns the user history as JSON format.
- [webhistory\\_index.html](#) is introduced to display “Your Activities” page.
- [webhistory\\_blueprint.py](#) is introduced and it has the following endpoints:
  - [index](#) : Constructs the “Your Activities” page.
  - [filteractivities](#) : Filter the activities according to the filter panel of the user interface.
  - [getmore](#) : Returns the most recent activities that are not displayed in “Your Activities” page.
- [webhistory\\_user\\_settings.py](#) is introduced to add “Your Activities” box to the user dashboard.
- [webhistory\\_config.py](#) is introduced. It keeps the following configurations:
  - [CFG\\_WEBHISTORY\\_MSGS](#) : Keeps the messages to display user activities.
  - [CFG\\_WEBHISTORY\\_DATEPICKER\\_DATETIME](#) : Format of the date picker of the filter in “Your Activities” page.
  - [CFG\\_WEBHISTORY\\_JSON\\_DATE\\_FORMAT](#) : Date format when using the dates of the activity in JSON objects.
  - [CFG\\_WEBHISTORY\\_JSON\\_TIME\\_FORMAT](#) : Time format when using the dates of the activity in JSON objects.
  - [CFG\\_WEBHISTORY\\_JSON\\_DATETIME\\_FORMAT](#) : Concatenation of the date format and time format.
- [CFG\\_WEBHISTORY\\_DATETIME\\_FORMAT](#) , which keeps the format of the datetime displayed in “Your Activities” page and box, is added to [invenio.conf](#)
- The type of the [date\\_creation](#) column of [UserQueryBasket](#) is changed from [Date](#) to [DateTime](#) .


- Two new columns `creation_date` and `action_code` are added into `UserBskBASKET` table. `webbasket_dblayer.py` is updated to accommodate these changes.

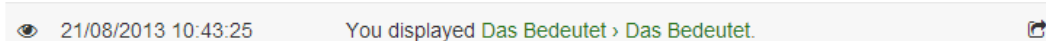
<b>Implemented By</b>	Şenar Postacı(SRDC)
-----------------------	---------------------

Table A.3: Implementation Description: RF2

<b>Feature ID</b>	RF3 (Repository Feature 3)
<b>Name</b>	“Share” option in “Your History” box
<b>Effort Spent</b>	Days
<b>Modules Affected / Created</b>	WebSession, WebMessage

### Description of the new feature

To share an activity, a tiny  icon is added near each activity in the “Your Activities” pages.



RF3 - A sample item on “Your Activities” page with sharing option

By clicking this icon, users may share their activities through a modal window. This modal window contains a form to send the activity to users.

### Implementation details

- `serialized` instance variable is added to `HistoryElement` to generate messages for shared activities.
- New methods are added to `HistoryCollector` to create and share user activities:
  - `serialize` (static): Serializes given database entry as list to use it for constructing message of the shared activity.
  - `get_share_message` (static): Converts serialized database entry to message.
- `share` endpoint is introduced in `webmessage_blueprint.py` to perform sharing an activity.
- `view` endpoint of `webmessage_blueprint.py` and `webmessage_view.html` are updated to display messages of shared activities.

- `getsharemessage` endpoint is introduced in `webhistory_blueprint.py` to get the message of given shared activity.
- Scripts of the `webmessage_add.html` is moved to `webmessage_common_js.html` to reuse them in `webhistory_index.html` while sharing an activity message.
- `hstSHARE` table is introduced to keep the shared activities. `webmessage_query.py` is also updated accordingly.
- `CFG_SHARE_MSGS` , which keeps the messages for shared activities, is added into `webhistory_config.py` .

<b>Implemented By</b>	Senan Postaci(SRDC)
-----------------------	---------------------

Table A.4: Implementation Description: RF3

<b>Feature ID</b>	RF4 (Repository Feature 4)
<b>Name</b>	Bibformat output templates to display blogs and blog posts differently
<b>Effort Spent</b>	3 Weeks
<b>Modules Affected / Created</b>	WebAccess, WebSearch, Bibformat

#### Description of the new feature

A different collection has been created to host the different types of records. The collection information is stored in the MARC tag 980\_a in Invenio. The possible values of the tag and the collection display names are the following:

Tag content	Collection name
BLOG	Blogs
POST	Posts
COMMENT	Comments
PAGE	Pages

There are BibFormat Templates defined for each type of record, as well as all the necessary BibFormat Elements. This way, the repository will follow the rules also defined to choose which BibFormat Templates to use for each record, depending on the collection they belong to.

### Implementation details

The file **democfgdata.sql** configures these collections by default when the demo site is created. In the same way, the BibFormat Templates, BibFormat Templates, and output format rules are configured by default.

<b>Implemented By</b>	Raquel Jiménez, Jaime García (CERN)
-----------------------	-------------------------------------

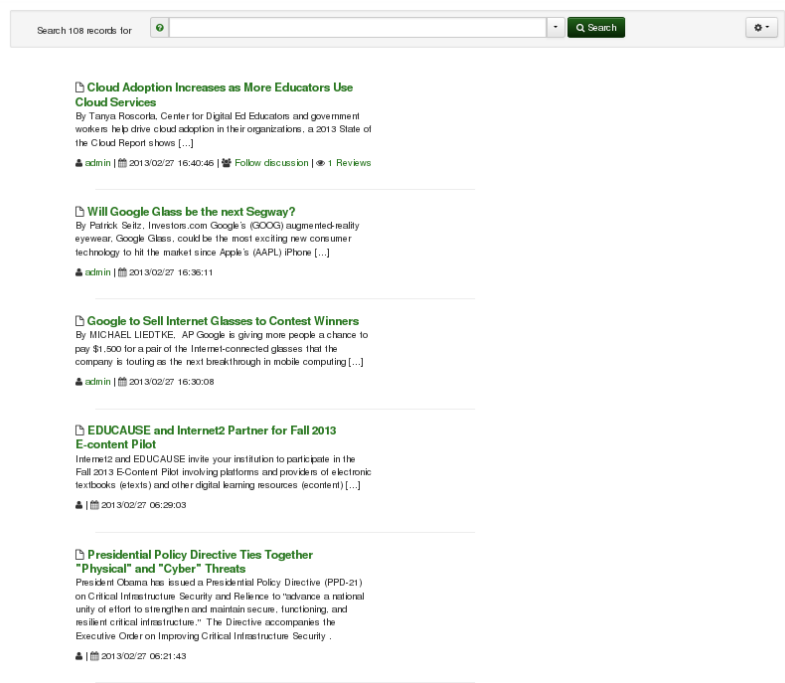
Table A.5: Implementation Description: RF4

<b>Feature ID</b>	RF6 (Repository Feature 6)
<b>Name</b>	Latest posts are displayed sorted by posted date
<b>Effort Spent</b>	1 Week
<b>Modules Affected / Created</b>	WebSearch, BibSort

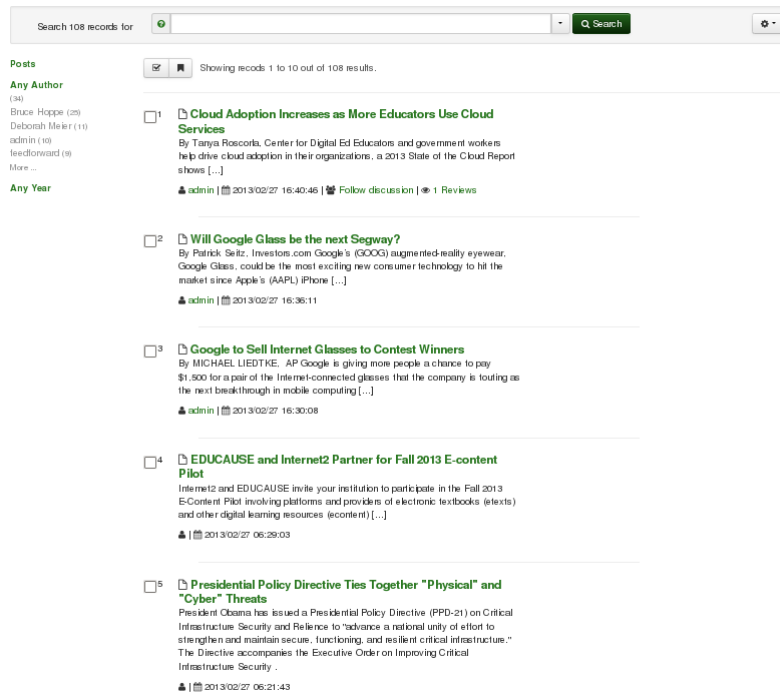
### Description of the new feature

Posts are always offered sorted chronologically by the date in which they were posted on their blog.

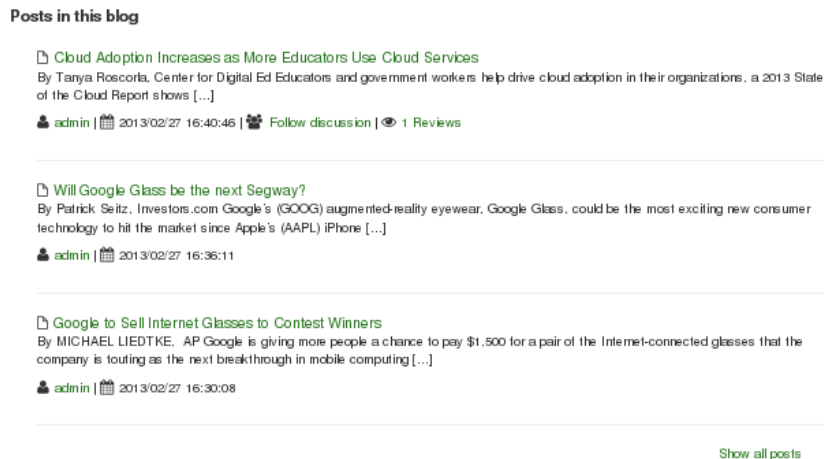
- Latest posts page: page displayed when the user clicks on Posts collection



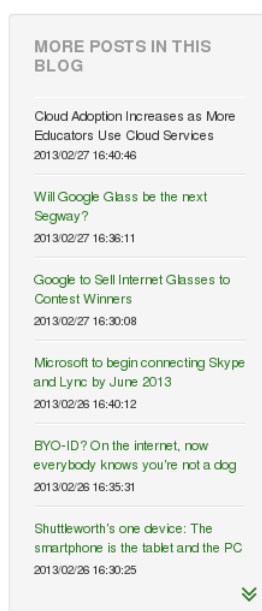
- Search posts results: results displayed when the user search within the Posts' collection



- “More posts in this blog” menu: menu offered within the post template with the rest of posts which belong to the same blog



- “Posts in this blog” section: section offered within the blog template with all the posts belonging to the same blog



### Implementation details

- A new sorting method called **posted date** has been added to the set of BibSort methods in order to sort records by the date given in the MARC tag 269\_c. Also a new washer called ( `sort_transform_format_date` has been implemented to transform the format of the date provided by the spider to the standard format `%Y/%m/%d %H:%M:%S`.
- A new property called **latest\_additions** has been added to the **Collection** class, which will contain the list of recids of each collection sorted by the selected sorting method.
- The blueprint **websearch\_blueprint.py** has been extended to propagate to the search the sorting parameters introduced to sort the latest additions.
- A new config variable **CFG\_WEBSEARCH\_INSTANT\_BROWSE\_AND\_SEARCH\_SAME\_SORTING** has been added in order to enable or disable the option to display the search results in the same order than the one selected to display the latest additions

Implemented By | CERN

Table A.6: Implementation Description: RF6

<b>Feature ID</b>	RF9 (Repository Feature 9)
<b>Name</b>	The archive stores and displays accordingly all record metadata received from the spider
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	BibUpload
<b>Description of the new feature</b>	
<p>The BibUpload module has been extended to allow that a script can run before and/or after the upload as a plugin. A pre-ingestion plugin has been developed to transform the metadata coming from the spider to the format that BibUpload and Invenio can understand.</p> <p>After the records have been inserted in the repository, the techniques described in RF4 and RF5 are used to display the metadata.</p>	
<b>Implementation details</b>	
<p>The bibupload command has been extended to accept extra arguments for pre- and post- ingestion plugins. In BlogForever, the command to be used to upload a new record coming from the spider would be 'bibupload batchupload -replace metadata_to_insert.xml -pre-plugin=bp_pre_ingestion -post-plugin=bp_post_ingestion'. The file <b>bp_pre_ingestion.py</b> will be run before the upload takes place, transforming the METS file coming from the spider into MARCXML. The METS content is parsed in order to extract the MARCXML that contains. The metadata is also enriched in several aspects:</p> <ul style="list-style-type: none"> <li>• FFT tags are inserted with references to every attached file downloaded from the spider.</li> <li>• Metadata tags are inserted linking each record to other existing records, like the parent record</li> <li>• The parent record license and visibility are propagated to the being uploaded record.</li> </ul>	
<b>Implemented By</b>	Raquel Jiménez, Jaime García (CERN)

Table A.7: Implementation Description: RF9



<b>Feature ID</b>	RF12 (Repository Feature 12)
<b>Name</b>	The archive can import METS
<b>Effort Spent</b>	1 Week
<b>Modules Affected / Created</b>	BibUpload
<b>Description of the new feature</b>	
<p>The archive is able to process METS files. For this propose, a pre-ingestion plugin is implemented to manage the METS files retrieved from the spider. One of the goals of this plugin is to parse the METS file, to extract the MARC living inside and to transform and to enrich it with the corresponding tags.</p>	
<b>Implementation details</b>	
<p>The pre-ingestion plugin is defined in <a href="#">modules/bibupload/lib/preprocess/bp_pre_ingestion.py</a> file. The Python module xml.dom.minidom is used to parse the given METS file. For more information about the pre-ingestion plugin see RF9.</p>	
<b>Implemented By</b>	Jaime García (CERN)

Table A.8: Implementation Description: RF12

<b>Feature ID</b>	RF14 (Repository Feature 14)
<b>Name</b>	Descriptive statistics are offered by record
<b>Effort Spent</b>	4 Days
<b>Modules Affected / Created</b>	WebSearch
<b>Description of the new feature</b>	
<p>Enabled the “statistics” tab in the detailed view of a record. The information offered is “People who viewed this page also viewed”, “People who downloaded files from this page also viewed” and “Download history graph”.</p>	

Implementation details	
<p>The logging of file downloads (<code>rnkDOWNLOADS</code>) and page views (<code>rnkPAGEVIEWS</code>) have been enabled.</p> <p>Using this information, the <code>record_usage.py</code> code is able to show “People who viewed this page also viewed”, “People who downloaded files from this page also viewed” and a “Download history graph” using the WebStat module.</p>	
<b>Implemented By</b>	CERN

Table A.9: Implementation Description: RF14

<b>Feature ID</b>	RF17 (Repository Feature 17)
<b>Name</b>	The archive displays a disclaimer about the originality of the content
<b>Effort Spent</b>	2 Days
<b>Modules Affected / Created</b>	BibFormat

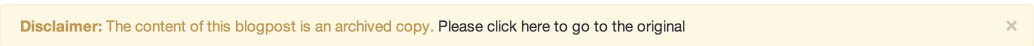
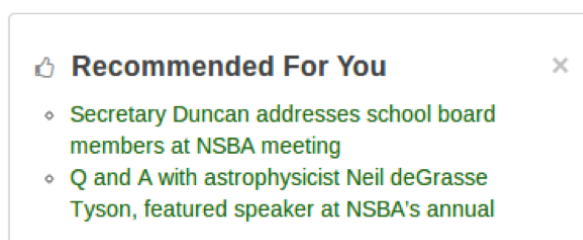
Description of the new feature	
<p>A disclaimer is displayed in every detailed record page saying that the corresponding blog, post, page or comment, is just an archived copy of the original one.</p>	
Implementation details	
<p>This is implemented in the <code>webstyle_template.py</code> file using HTML and Python. The disclaimer says that the presented content is just an archived copy, not the original. A link to the original element is also offered.</p>	
	
RF17 - Disclaimer	
<b>Implemented By</b>	Raquel Jiménez Encinar (CERN)

Table A.10: Implementation Description: RF17

<b>Feature ID</b>	RF22 (Repository Feature 22)
<b>Name</b>	“Your Preferences” box as part of the user dashboard
<b>Effort Spent</b>	1 Week
<b>Modules Affected / Created</b>	WebSearch, WebSession

### Description of the new feature

A new box containing recommended records based on user search criteria has been added into “Your Account” page.



### Implementation details

- Two new database tables, `query_term` and `user_query_term` are introduced. Moreover, `log_query_terms` function is added into `search_engine.py` to keep history of the search terms used by each user.
- `webrecommend.py` module that contains functions to recommend the records to the users are developed. These functions are:
  - `get_unread_records` : Returns the record ID's that are not viewed by the user.
  - `get_query_terms` : Returns the list of the query terms that have been used by the user
  - `get_recommended_content` : Returns the unread records based on word similarity with the query terms used by the user.
- Three configuration parameters have been added into `websession_config.py` :
  - `CFG_RECOMMENDATION_RANK_METHOD` : The name of the word similarity ranking method
  - `CFG_RECOMMENDED_CONTENT_NUMBER` : The number of records recommended in “Your Account” page.
  - `CFG_MOST_FREQUENT_TERM_NUMBER` : The number of most frequent terms considered in recommendation.

<ul style="list-style-type: none"> <li>• <b>webrecommend_user_settings.py</b> is introduced to add “Recommended for you” box to “Your account” page.</li> </ul>
<b>Implemented By</b>   SRDC

Table A.11: Implementation Description: RF22

<b>Feature ID</b>	RF23 (Repository Feature 23)
<b>Name</b>	The archive stores the comments of blog posts and displays them as part of the blog posts
<b>Effort Spent</b>	3 Days
<b>Modules Affected / Created</b>	BibFormat
<b>Description of the new feature</b>	
<p>Comments of blog posts are displayed to the user together with the specific blog post. The two latest comments are displayed by default. If the user wants to see all the comments needs to click on the “Show all comments” link.</p>	
<b>Implementation details</b>	
<p>A new BibFormat element called “bfe_post_comments” is created. HTML, JavaScript and Python is used. This element is used in the BibFormat template “PostHTML.bft”.</p>	
<b>Implemented By</b>	Raquel Jiménez Encinar (CERN)

Table A.12: Implementation Description: RF23

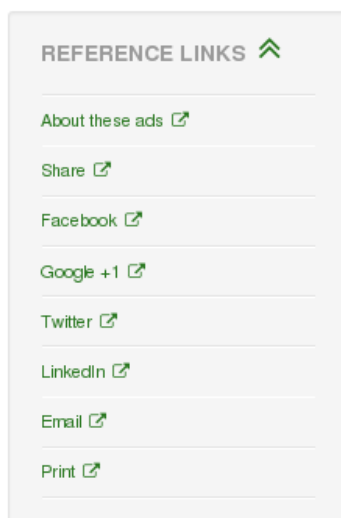
<b>Feature ID</b>	RF24 (Repository Feature 24)
<b>Name</b>	Links to other sources within the blog posts and comments are displayed separately
<b>Effort Spent</b>	1 Week
<b>Modules Affected / Created</b>	WebBlog

### Description of the new feature

The repository displays in the detailed record page of a blog post a menu with all the links used as references. If any of these links is pointing to a content already stored into the archive, a link to the corresponding record is offered. Reference links can be either provided by the spider or is the repository who extracts them in case the spider does not provide them.

### Implementation details

A new BibFormat element called “BFT\_LINKS\_MENU” is created for this propose, which is used in the BibFormat template “PostHTML.bft”. This element also displays the link to the archived content in case the reference link is pointing to a content already stored in the archive.



<b>Implemented By</b>	Raquel Jiménez Encinar (CERN)
-----------------------	-------------------------------

Table A.13: Implementation Description: RF24

<b>Feature ID</b>	RF25 (Repository Feature 25)
<b>Name</b>	The archive displays the tags of a blogs and blog posts
<b>Effort Spent</b>	3 Days
<b>Modules Affected / Created</b>	BibFormat

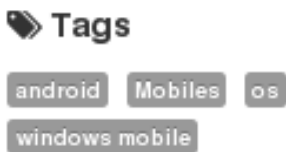
<b>Description of the new feature</b>	
<p>Tags associated with blogs and blog posts are displayed in the detailed record page as links in such way that a search by the corresponding tag is triggered by clicking on it. Tags are provided by the spider and the repository get and display them using a BibFormat element.</p>	
<b>Implementation details</b>	
<p>The new BibFormat element “BFT_TAGS” is created which is used in the BibFormat templates “BlogHTML.bft”, “PostHTML.bft”</p>	
	
<b>Implemented By</b>	Raquel Jiménez Encinar (CERN)

Table A.14: Implementation Description: RF25

<b>Feature ID</b>	RF26 (Repository Feature 26)
<b>Name</b>	BlogUploader command line to upload, update and delete a list of blogs
<b>Effort Spent</b>	3 Weeks
<b>Modules Affected / Created</b>	WebBlog
<b>Description of the new feature</b>	
<p>In order to let administrators to edit a list of blogs and to insert, delete or update them into the repository, a new command line tool has been implemented.</p>	
<b>Implementation details</b>	
<p>A new command line tool called “bloguploader” is implemented. The following modes are offered:</p>	

- Insert a blog (-i, -blog\_insert): This option let admins insert a list of blogs in the archive. Each blog is represented by its url, title (optional), topic and license.  
E.g: <http://blogforever.eu>,BlogForever,topic1,license1  
<http://blogs.physicstoday.org/>,,topic1,license3
- Delete a blog (-d, -blog\_delete): This option let admins delete a list of blogs from the archive. Each blog is represented just by its url. E.g:  
<http://blogforever.eu>  
<http://blogs.physicstoday.org/>
- Update a blog (-U, -blog\_update): This option let admins update a list of blogs in the archive. Each blog is represented by its url, title (optional), topic and license.  
E.g: <http://blogforever.eu>,BlogForever,topic2,license2  
<http://blogs.physicstoday.org/>,Physicstoday,topic1,license2

The input file is a CSV file where the elements of each row (blog elements) are separated by commas. The output file is a marxml file that contains all the records to be inserted, deleted or updated by BibUpload.

<b>Implemented By</b>	Raquel Jiménez Encinar (CERN)
-----------------------	-------------------------------

Table A.15: Implementation Description: RF26

<b>Feature ID</b>	RF28 (Repository Feature 28)
<b>Name</b>	The archive displays the author of blog posts and comments
<b>Effort Spent</b>	3 Days
<b>Modules Affected / Created</b>	BibFormat

#### Description of the new feature

The author of blog posts and comments are displayed with them as a link in such way that a search by author is triggered by clicking on it. The author is displayed in both, the brief record and the detailed record.

#### Implementation details

The BibFormat templates “Post\_HTML\_brief.bft” and “Comment\_HTML\_brief.bft” are enriched with the BibFormat element “BFE\_AUTHORS”. On the other hand, the BibFormat templates “PostHTML.bft” and “CommentHTML.bft” are enriched with the new

BibFormat elements “BFE_POST_AUTHOR” and “BFE_COMMENT_AUTHOR” respectively.	
<b>Implemented By</b>	Raquel Jiménez Encinar (CERN)

Table A.16: Implementation Description: RF28

<b>Feature ID</b>	RF29 (Repository Feature 29)
<b>Name</b>	The archive alerts the user when there are software updates
<b>Effort Spent</b>	Days
<b>Modules Affected / Created</b>	WebSession
<p><b>Description of the new feature</b></p> <p>The archive checks and informs the administrators via user dashboard when there are software updates.</p> <p>A newer version of Invenio is available for download. You may want to visit <a href="http://invenio-software.org/wiki/Installation/Download">http://invenio-software.org/wiki/Installation/Download</a></p> <p>RF29 - The alert displayed on the user dashboard when there are software updates</p> <p><b>Implementation details</b></p> <ul style="list-style-type: none"> <li>The <code>index</code> endpoint of <code>webaccount.blueprint.py</code> is modified to check if there are any software updates.</li> </ul>	
<b>Implemented By</b>	Alper Çınar (SRDC)

Table A.17: Implementation Description: RF29

<b>Feature ID</b>	RF31 (Repository Feature 31)
<b>Name</b>	The archive offers a complete blog submission interface to submit, modify and delete blogs/posts
<b>Effort Spent</b>	1 Month
<b>Modules Affected / Created</b>	WebSubmit, WebBlog



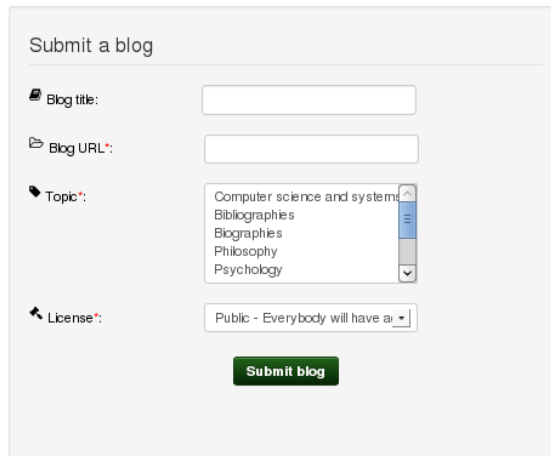
## Description of the new feature

The archive offers a complete submission interface to let users and admins to submit new blogs, to modify certain specific metadata of a blog, and to delete either a blog (as result all its comments and blog posts will be deleted) or a single blog post.

## Implementation details

To offer a complete blog submission interface, two new document types has been created:

- Blog Submission(BSI), which will be used by administrators. The developed actions are:
  - Submit a Blog: form to let admins submit a blog. Users should provide the URL of the blog, title, license (see RF53) and topic. The blog is submitted directly.



Submit a blog

Blog title:

Blog URL\*:

Topic\*:   
Bibliographies  
Biographies  
Philosophy  
Psychology

License\*:

- Modify a Blog: form to let admins modify specific metadata of a blog. Users should provide the URL of the blog and select the field/s they want to modify. The blog is modified directly.

- Delete a Blog: form to let admins delete a blog and all its descendants. Users should provide the URL of the blog. The blog and all its descendants are deleted directly.

- Delete a Post: form to let admins delete a post. Users should provide the URL of the post. The post is deleted directly.

- Blog Submission (Refereed)(BSIREF), which will be used by users and referees. The developed actions are:
  - Submit a Blog: form to let users submit a blog. Users should provide the URL of the blog, title, license (see RF53) and topic. Users should wait for the referee's decision.
  - Approve Blog Submission: form to let referees approve or reject a submitted blog.
  - Modify a Blog: form to let users modify specific metadata of a blog. Users should provide the URL of the blog and select the field/s they want to modify. Users should wait for the referee's decision.

- Approve Blog Modification: form to let referees approve or reject modifications on the metadata of a blog.
- Delete a Blog: form to let users delete a blog and all its descendants. Users should provide the URL of the blog. Users should wait for the referee’s decision.
- Approve Blog Deletion: form to let referees approve or reject the deletion of a blog.
- Delete a Post: form to let users delete a post. Users should provide the URL of the post. Users should wait for the referee’s decision.
- Approve Post Deletion: form to let referees approve or reject the deletion of a post.

New websubmit functions have been also implemented in order to customize the submission. Functions to send e-mails or to display messages to the user after every action, functions to send e-mails to the referee giving him the option to reject or approve an action, function to check the validation of a particular URL, function to carry out the deletion of blogs and/or posts. All the defined functions are: `APM_Mail_Final_Decision_to_User`, `APM_Print_Success`, `APO_Mail_Final_Decision_to_User`, `APO_Print_Success`, `APP_Mail_Final_Decision_to_User`, `APP_Print_Success`, `APS_Mail_Final_Decision_to_User`, `APS_Print_Success`, `DBI_Mail_Approval_Request_to_Referee`, `DBI_Mail_Blog_Deleted_to_User`, `DBI_Mail_Notification_to_User`, `DBI_Print_Success`, `DPI_Mail_Approval_Request_to_Referee`, `DPI_Mail_Notification_to_User`, `DPI_Mail_Post_Deleted_to_User`, `DPI_Print_Success`, `MBI_Mail_Approval_Request_to_Referee`, `MBI_Mail_Blog_Modified_to_User`, `MBI_Mail_Notification_to_User`, `MBI_Print_Success`, `SBI_Mail_Approval_Request_to_Referee`, `SBI_Mail_Blog_Submitted_to_User`, `SBI_Mail_Notification_to_User`, `SBI_Print_Success`, `Make_Delete_Records`, `Check_URL`.

The new forms have been created through the WebSubmit admin interface. Once this is done, all the code that has been written is dumped into a file and the file `democfgdata.sql` is enriched with that code.

In addition of this, two options are offered in the detailed record page to delete or to modify a record directly. These actions are “Ask for Deletion” and “Ask for Modification”. In order to implement this, two new BibFormat elements have been created: “`bfe_ask_for_deletion`” and “`bfe_ask_for_modification`”. By clicking on these options the user is redirected to the WebSubmit interface to performs the wished action. It goes to BSI if the user is admin, otherwise it goes to BSIREF.

In order to manage submissions, two new restricted and hidden collections have been created:

- Provisional Blogs: contains all the submitted (approved) blogs
- Rejected Blogs: contains the blogs rejected by the referee

On the other hand, in order to manage the restrictions on the collections mentioned above the file `access_control_config.py` has been amended creating new roles, new authorizations and new restrictions.

<b>Implemented By</b>	Raquel Jiménez Encinar (CERN)
-----------------------	-------------------------------

Table A.18: Implementation Description: RF31

<b>Feature ID</b>	RF32 (Repository Feature 32)
<b>Name</b>	Users are able to remove their personal data
<b>Effort Spent</b>	3 Days
<b>Modules Affected / Created</b>	WebSession, WebAccess

### Description of the new feature

A user is able to disable his/her account. This service can be accessed from `https://<site-address>/youraccount/edit`. Before confirmation, the user is able to select an option to delete the personal data completely or keep it in the database. If the former one is selected, all the user information except the shared data related with groups, public baskets and messages is deleted from database. The site administrator is able to enable/disable this selection.

To make sure that only authorized users are performing this operation, an additional password affirmation is also needed.

If you want to deactivate your account, please fill the form below. To remove your account completely, also enable removing your personal data.

Remove my personal data

Password:

Note: Please confirm your password to continue.

[Deactivate now](#)

If the user data is kept, the user is able to reactivate his/her account by signing in with same email/nickname and password. If the user tries to register with same credentials, s/he is encouraged to log in to reactivate his/her account.

### Registration failure

It looks like you have deactivated your account. To reactivate your account, please log in with your email/nickname and password. [Login](#)

When the user reactivates his/her account, a “welcome message” appears and the user can access his/her personal data.

## Login

Welcome back. Your account has been reactivated successfully.

[Go to Your Account page](#)

### Implementation details

- Since password confirmation is not possible for external login method, users using external login are not able to remove their accounts until they get new password.
- **CFG\_ACCESS\_CONTROL\_ENABLE\_SUSPENDED\_ACCOUNTS** parameter has been added in **invenio.conf** . It defines whether the user accounts can be suspended or not. If this is not set to 1, deactivation option is not provided.
- To distinguish suspended users from other user types, value 3 is set to **note** column of the **user** table in database.
- In **webuser.py** , **remove\_user** and **deactivate\_user** functions which handle database transactions have been implemented.
- To find tables with user data, tables with column **id\_user** or **uid** are queried in **remove\_user** function. The query can be extended by adding new column names to the query list in the same function.
- New message with key **21** that is related to reactivation has been inserted to **CFG\_WEBACCESS\_WARNING\_MSGS** dictionary in **access\_control\_config.py** .
- **loginUser** and **registerUser** functions in **webuser.py** have been modified.
- In **websession\_webinterface.py** , **delete** method has been reimplemented, **login** and **register** functions have been edited.
- In **webaccount.py** , **perform\_delete** function has been reimplemented.
- In **websession\_templates.py** , **tmpl\_user\_preferences** and **tmpl\_account\_delete** functions have been modified.

<b>Implemented By</b>	Şenan Postacı(SRDC)
-----------------------	---------------------

Table A.19: Implementation Description: RF32

<b>Feature ID</b>	RF34 (Repository Feature 34)
<b>Name</b>	The archive displays and suggests similar records to the user
<b>Effort Spent</b>	1 Week
<b>Modules Affected / Created</b>	WebSearch, WebInterface
<b>Description of the new feature</b>	
<p>Extended Invenio's record similarity functionality to consider blog records as a special case. To determine the similarity of two blogs, the level of similarity between their posts is first calculated and then aggregated to come up with a score.</p>	
<b>Implementation details</b>	
<p>Added an function step before <code>blog_extend.bibrank</code> in <code>search_engine.py</code> called <code>blog_extend.bibrank</code>. Given one blog, it will get all the posts, get the similar posts for all of them, then aggregate the results to get the most similar posts for the blog in general, and finally group them by parent blog. The final outcome is a list of blogs similar to the blog given as an argument.</p>	
<b>Implemented By</b>	CERN

Table A.20: Implementation Description: RF34

<b>Feature ID</b>	RF35 (Repository Feature 35)
<b>Name</b>	The archive displays other blogs that were viewed by people who also viewed the current blog
<b>Effort Spent</b>	2 Days
<b>Modules Affected / Created</b>	BibRank
<b>Description of the new feature</b>	
<p>The repository displays in the tab "Usage statistics" of the detailed blog record page, the list of blogs that were viewed by people who also viewed the current blog. Each viewed blog will be showed with the number of different people who had viewed it.</p>	

### Implementation details

The query located in the file `bibrank_downloads_similarity.py` in the function “`calculate_reading_similarity_list()`” is amended grouping the records by blog collection, in order to get all the blogs that were viewed by people who also read a specific blog. The final result offers the name of the blogs that were viewed by people who also viewed the current blog with the number of users who viewed each of those blogs.

<b>Implemented By</b>	Raquel Jiménez Encinar (CERN)
-----------------------	-------------------------------

Table A.21: Implementation Description: RF35

<b>Feature ID</b>	RF36 (Repository Feature 36)
<b>Name</b>	The archive identifies and stores the topic of blogs and blog posts to let users navigate through the archive by topic
<b>Effort Spent</b>	2 Days
<b>Modules Affected / Created</b>	WebSubmit, WebBlog

### Description of the new feature

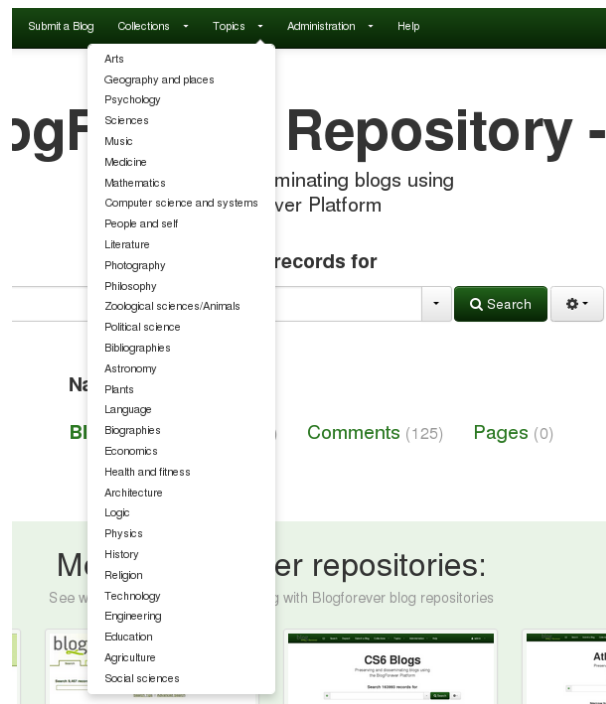
At submission time users can select the topic/s the blog belongs to. After this, at pre-ingestion time, the selected topic/s are propagated to the descendants of the blog, so if users search for a specific topic, the search will return the blog and also all its descendants.

### Implementation details

A new config variable has been added named `CFG_BLOG_TOPICS` which define the list of topics that the repository offers to users. This list can be edit by the administrator.

A new field has been added to the submission form as a multiple select to be able to select more than one topic.

The user can navigate through the topics by clicking on the drop menu displayed in the main page,



or also by clicking on the tags displayed with each record.



Implemented By | CERN

Table A.22: Implementation Description: RF36



<b>Feature ID</b>	RF40 (Repository Feature 40)
<b>Name</b>	The archive validates the content received from the spider
<b>Effort Spent</b>	1 Day
<b>Modules Affected / Created</b>	BibUpload, BibIngest
<b>Description of the new feature</b>	
<p>The module BibIngest described in RF87 provides a method that calculates the md5 hash of every file fetched from the spider and compares it with the md5 hash provided by the spider. This method is used in <a href="#">bp_pre_ingestion.py</a> in the pre-ingestion processing described in RF9.</p>	
<b>Implementation details</b>	
<p>See BibIngest module described in RF87.</p>	
<b>Implemented By</b>	Nikolaos Kasioumis (CERN)

Table A.23: Implementation Description: RF40

<b>Feature ID</b>	RF41 (Repository Feature 41)
<b>Name</b>	The archive detects and eliminates spam content
<b>Effort Spent</b>	3 Weeks
<b>Modules Affected / Created</b>	BibSpam, BibSched
<b>Description of the new feature</b>	
<p>BibSpam daemon is scheduled to run periodically and check all repository records to identify spam content. Spam records are flagged with a special MARC tag 911_s as spam with “SPAM” label.</p>	
<b>Implementation details</b>	
<p>BibSpam is based on URL blacklists such as SpamHaus <sup>a</sup> to identify spam.</p> <p>The operation of the BibSpam module can be summarized as follows:</p>	

1. The tasklet named `bst_spam_detection.py`, iterates over the list of records the user provides, or over the new records that were ingested into the repository since the last execution of the BibSpam tasklet.
2. For each of these records, it checks if metadata element `520_u` exists (URL). If not the record is skipped because the spam classification is performed based on the record URL.
3. The spam classifier checks if the URL is spam. If so, The word “SPAM” is written in `911_u` tag.

After the daemon process has been completed, the admin should run BibIndex and WebColl to see the changes in the records.

Configuration file: **`etc/BibSpam/BibSpam.cfg`**

Command line execution: **`sudo -u www-data /opt/invenio/bin/BibSpam`**

<sup>a</sup><http://www.spamhaus.org>

<b>Implemented By</b>	Vangelis Banos (AUTH)
-----------------------	-----------------------

Table A.24: Implementation Description: RF41

<b>Feature ID</b>	RF45 (Repository Feature 45)
<b>Name</b>	The archive is able to inter-operate with federated search engine dbwiz (SRU Server)
<b>Effort Spent</b>	3 Weeks
<b>Modules Affected / Created</b>	BibFormat, WebSearch, WebStyle

#### Description of the new feature

SRU is a standard XML-focused search protocol for Internet search queries. Support for the SRU protocol has been added in Invenio.

#### Implementation details

Any 3<sup>rd</sup> party software or web user can perform http requests in the SRU server implemented at `/sru` URL endpoint. Results are formatted using XML and the specific SRU schemas described in <http://www.loc.gov/standards/sru/resources/schemas.html>.

Example request: <http://bf3.itc.auth.gr/sru?version=1.1&operation=searchRetrieve&query=information&maximumRecords=10&recordSchema=dc>

SRU 1.2 service. parameters:

- operation → (searchRetrieve, explain, scan, CQL)
- version → only 1.2 is supported
- query (search query)
- startRecord (int)
- maximumRecords (int)
- recordPacking (xml is default, other is string)
- recordSchema  
→ <http://www.loc.gov/standards/sru/resources/schemas.html>
- resultSetTTL → not supported
- stylesheet → Reference: <http://www.loc.gov/standards/sru/specs/common.html#stylesheet>
- extraRequestData → not supported

scan and CQL are not supported yet

<b>Implemented By</b>	Apostolos Papadopoulos (ALTEC), Vangelis Banos (AUTH)
-----------------------	---

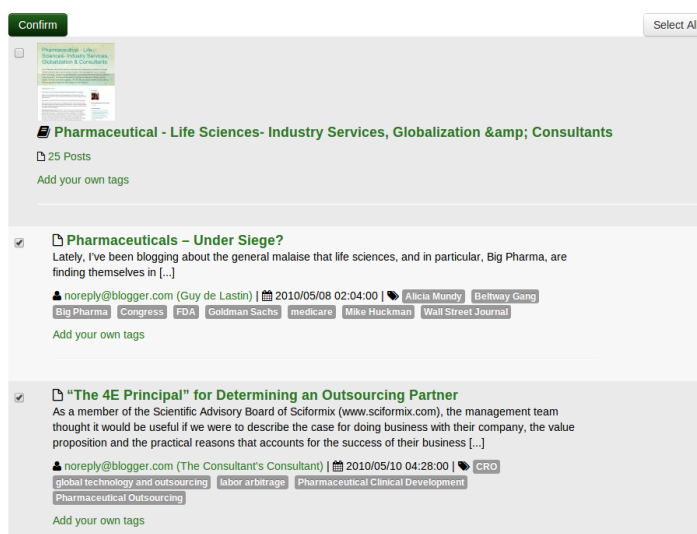
Table A.25: Implementation Description: RF45

<b>Feature ID</b>	RF46 (Repository Feature 46)
<b>Name</b>	Users can create personal collections of their favourite blogs
<b>Effort Spent</b>	Days
<b>Modules Affected / Created</b>	WebBasket

#### Description of the new feature

With this feature blogs can be added to user personal baskets with their blog posts.

When deleting a blog from a personal basket, user is asked to select which blog posts of that blog s/he also wants to delete. When deleting a blog post from a personal basket, checks if the blog of that blog post is also in the same basket. If so, user is asked to select which blog posts of that blog s/he also wants to delete.



## Implementation details

- `perform_request_add` function of `webbasket.py` is modified to add the blog posts of the blog when the user adds a blog to personal basket.
- `modify` method of `WebInterfaceYourBasketsPages` class is modified to check if there are any other records user may want to delete when removing a blog or blogpost from a personal basket.
- `perform_request_confirm_delete` is added to `webbasket.py` to display the list of related blogs and blog posts with the selected record.
- `tmpl_delete_related_record_confirmation` method is introduced in `Template` class of `WebBasket` module, which constructs the page containing the list of the records which the user may also want to remove.
- `confirm_delete` method is introduced in `WebInterfaceYourBasketsPages` class, which deletes the selected blogs and blog posts.
- `get_basket_recids` is added to `webbasket_dblayer.py` that returns the list of the record IDs of given basket.
- Two new functions are added into `webblog_utils.py` :
  - `extend_with_blog_posts` : Extends the list of blog IDs with their blog post IDs.
  - `get_related_records_in_basket` : Returns the IDs of the blog and blog posts related with the given record from the given basket.

Implemented By	Alper Çınar (SRDC)
----------------	--------------------

Table A.26: Implementation Description: RF46

<b>Feature ID</b>	RF47 (Repository Feature 47)
<b>Name</b>	Description of how to cite archived records is presented prominently with each record
<b>Effort Spent</b>	3 Days
<b>Modules Affected / Created</b>	BibFormat

### Description of the new feature

A user needs to link and cite the content of the archive. Therefore, the way how to link and how to cite a record is presented prominently in the detailed view of the record by a format element.

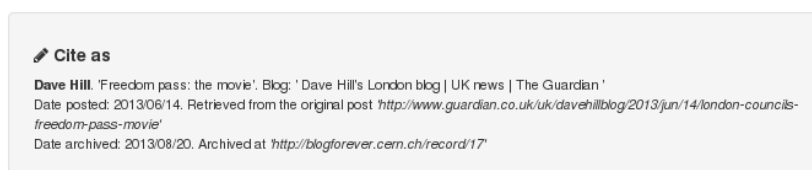
### Implementation details

A new format element called **bfe\_citation\_box.py** is created. This element displays the description of how users should cite any content in the archive. This description includes:

- For blogs:  
“title”  
Retrieved from the original blog “original\_url”  
Date archived: (record\_creation\_date). Archived at “record\_url”



- For blog posts:  
author. “title”. Blog: “blog\_title”  
Date posted: (record\_posted\_date). Retrieved from the original post “original\_url”  
Date archived: (record\_creation\_date). Archived at “record\_url”



- For comments:  
 author. Blog post: “blog\_title” Retrieved from the original comment  
 “original\_url”  
 Date archived: record\_creation\_date). Archived at “record\_url”



<b>Implemented By</b>	Raquel Jiménez Encinar (CERN)
-----------------------	-------------------------------

Table A.27: Implementation Description: RF47

<b>Feature ID</b>	RF48 (Repository Feature 48)
<b>Name</b>	The archive provides the option to translate its content on demand
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	BibFormat, WebComment, WebMessage, WebStyle

**Description of the new feature**

This feature enables users to translate the content of records, messages, reviews and comments. The language used by the user determines the language of the translation.



The user can translate the corresponding part by clicking “Translate” link over the context. Moreover, the text of the “Translate” link appears in the language of the user. In the sample below, the user using the platform in English translates the message in German to English.

**From** admin  
**Subject** Translate  
**Sent on** 2013-05-20 17:31:25  
**Sent to** admin  
**Sent to groups**

English ▾

**Translate**

Dies ist die Botschaft, werde übersetzt werden sollen.

The user is able to undo the translation by clicking “Show Original” link.

**From** admin  
**Subject** Translate  
**Sent on** 2013-05-20 17:31:25  
**Sent to** admin  
**Sent to groups**

English ▾

**Show original**

This is the message that will be translated.

### Implementation details

- Google translate gadget has been utilized for translation task.
- “Translate” link is added over the records, messages, reviews and comments to translate the content.
- To add “Translate” link, [webcomment\\_comments.html](#) , [webcomment\\_reviews.html](#) , [webmessage\\_view.html](#) and some of the bibformat templates are modified, also a new bibformat element, [bfe\\_translate.py](#) , is created.

<b>Implemented By</b>	SRDC
-----------------------	------

Table A.28: Implementation Description: RF48

<b>Feature ID</b>	RF52 (Repository Feature 52)
<b>Name</b>	Users can tag archived records with personal tags
<b>Effort Spent</b>	3 Weeks
<b>Modules Affected / Created</b>	WebTag

## Description of the new feature

Users are able to assign personal tags to the records. With these tags, the user is able to organize the records in personal collections according to their preferences.

Users are able to create and delete tags, as well as attach and detach them from records.

## Implementation details

The functionality is provided by a new module called WebTag. The module utilizes infrastructure provided by Invenio Next.

### Database Backend

WebTag defines a set of classes which are translated into database tables using SQL Alchemy ORM:

- **WtgTAG** : A named tag object owned by a user. The tag's name cannot contain non-alphanumeric characters.
- **WtgTAGRecord** : Association between a WtgTAG and a record (class **Bibrec** ).
- **WtgTAGUsergroup** : Describes the permissions given by the owner to a group of users. This mechanism is not yet implemented, but we plan adding group and public tags later.

### User Interface

WebTag defines an **InvenioBlueprint** . The user interface consists of:

- Tag Cloud mode of viewing tags (</yourtags/display/cloud>)

#### Tag Cloud



asshole fff fun interesting Qua Quo  
to read to share work

- Table mode of viewing tags (</yourtags/display/list>): this view allows users to create and delete tags as well as display them ordered by different parameters.



## Your tags

### Tag list

<input type="checkbox"/>	Tag	Documents	Action
<input type="checkbox"/>	asshole	1	Delete
<input type="checkbox"/>	fff	1	Delete
<input type="checkbox"/>	fun	1	Delete
<input type="checkbox"/>	interesting	3	Delete
<input type="checkbox"/>	Qua	0	Delete
<input type="checkbox"/>	Quo	0	Delete
<input type="checkbox"/>	to read	3	Delete
<input type="checkbox"/>	to share	1	Delete
<input type="checkbox"/>	work	0	Delete

Delete Selected

New tag

- List of tags in document search and detailed view: for logged-in users, a list of tags is displayed when viewing a document.



- Tag editor: in document view, users can open the editor to attach and detach tags from a record. It can also create new tags and attach them. The editor uses javascript and commits all changes immediately to the server by AJAX requests.
- List of records associated with a tag ([/yourtags/tag/<id\\_tag>/records](/yourtags/tag/<id_tag>/records)): Displays all records associated with the selected tag.
- User Settings widget in Your Account page presents statistics about tags as well as links to other parts of the module's user interface.

All information sent to the server is validated using WTForms validators in order to maintain name restrictions and check users' permissions.

The list of tags in document view is implemented with the template context function `bfm_webtag_record_tags`. The function can be used in various templates which allows easy customization of application interface.

### Configuration

The configuration variables are located inside `webtag-config.py`:

- **CFG\_WEBTAG\_NAME\_MAX\_LENGTH** : maximum length of tag name.
- **CFG\_WEBTAG\_NAME\_REPLACEMENTS\_SILENT** : list of regular expression replacements applied before saving a tag name.
- **CFG\_WEBTAG\_NAME\_REPLACEMENTS\_BLOCKING** : list of regular expression replacements applied on a tag name. If a match is found, the name is considered invalid. The expressions are used to suggest a similar valid name.
- **CFG\_WEBTAG\_ACCESS\_NAMES** and **CFG\_WEBTAG\_ACCESS\_RIGHTS** : permission levels for group and public tags. Currently not used.

### File structure

The module's files:

- **lib/\*** : Application logic in Python and Javascript.
- **etc/templates/\*** : Jinja2 templates for generating HTMLs. Files without **\_base.html** suffix are provided to simplify interface customizations.

<b>Implemented By</b>	CERN
-----------------------	------

Table A.29: Implementation Description: RF52

<b>Feature ID</b>	RF53 (Repository Feature 53)
<b>Name</b>	The archive respects content licenses and displays useful information about them
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	WebSubmit, WebSearch, WebAccess

### Description of the new feature

This feature can be split in 2 parts:

- In order to display the license information captured by the spider, a BibFormat Element has been created that displays this information, and that element has been used in the convenient BibFormat Templates.
- The administrators (or any user allowed to do it) submit a new URL to be crawled using WebSubmit are asked for the visibility that the Blog (and all the child records: posts, comments and pages) should have. The three options are:

- Public - Everybody will have access to the content.
- Restricted - Only registered users will have access to the content.
- Private - Only you will have access to the content.

These visibility options are propagated to the child records when they are fetched from the spider (see RF9 for more details on pre-ingestion processing). This information is stored in the MARC metadata in the field 980 (collection) and used afterwards in the WebAccess configuration, allowing the access to the content to the appropriate users.

### Implementation details

See RF31 for more details on WebSubmit. The files **democfgdata.sql** and **access\_control.config.py** contain the default configuration that will be installed, and the following collections RESTRICTEDCONTENT and PRIVATECONTENT are included. If the content is not tagged with one of these values in the 980 MARC tag it is considered to be Public.

<b>Implemented By</b>	Raquel Jiménez, Jaime García (CERN)
-----------------------	-------------------------------------

Table A.30: Implementation Description: RF53

<b>Feature ID</b>	RF54 (Repository Feature 54)
<b>Name</b>	The archive keeps all the different versions of a record
<b>Effort Spent</b>	1 Week
<b>Modules Affected / Created</b>	BibIngest

### Description of the new feature

Versioning is enabled for every data object stored in the digital repository. When any data object is modified, a new version of it is kept in the ingestion database mongoDB. Therefore, the repository stores all the different versions of all the data objects.

### Implementation details

A new parameter called “version” is added to the usage settings of the module BibIngest, which is keeping the last version of each record. A new config variable called “CFG\_BIBINGEST\_VERSIONING” is also added in order to

manage versions, if True, whenever an ingestion package is updated old versions are kept.
<b>Implemented By</b>   Nikolaos Kasioumis (CERN)

Table A.31: Implementation Description: RF54

<b>Feature ID</b>	RF57-58-61 (Repository Feature 57-58-61)
<b>Name</b>	The archive provides a ranking method based on the user rating of content (RF57) A user can rank archived content based on specific users' content rating (RF58) The archive ranks blogs based on their views and downloads (RF61)
<b>Effort Spent</b>	1 Week
<b>Modules Affected / Created</b>	BibRank

#### Description of the new feature

These three features have been implemented in the same development branch and add two different ranking method templates to the existing ones: “Number of Record Views” (record.view) and “Average Review Score” (average\_score).

“Number of Record Views” ranking method calculates the number of the visits to the record. Each visit to the same record is counted as 1 for the visits occurred within a minute.

“Average Review Score” calculates the average scores of the records.

#### Implementation details

- 2 new ranking templates are added: `template_average_score.cfg` and `template_record_view.cfg`
- `template_record_view.cfg` has a parameter `time_interval` that decides the interval to delete consequent record views, i.e., it does not matter how many times a user views a record in a given time interval, it is counted only once.
- 4 new functions have been implemented in `bibrank_tag_based_indexer.py` :

<ul style="list-style-type: none"> <li>– <b>record_view</b> : executes <b>bibrank_engine</b> method for <b>record_view</b> ranking method</li> <li>– <b>record_view_exec</b> : Ranks total number of record visits without checking the user IP</li> <li>– <b>average_score</b> : executes <b>bibrank_engine</b> method for <b>average_score</b> ranking method</li> <li>– <b>average_score_exec</b> : Ranks average review score for records</li> <li>• 2 new files have been created: <b>bibrank_record_view_indexer.py</b> and <b>bibrank_average_score_indexer.py</b> <ul style="list-style-type: none"> <li>– <b>bibrank_record_view_indexer</b> contains the functions that are used for indexing visit counts of each record.</li> <li>– <b>bibrank_average_score_indexer</b> contains the function that are used for indexing average review score of each record.</li> </ul> </li> </ul>		
<table border="1" style="width: 100%;"> <tr> <td style="width: 20%;"><b>Implemented By</b></td> <td>SRDC</td> </tr> </table>	<b>Implemented By</b>	SRDC
<b>Implemented By</b>	SRDC	

Table A.32: Implementation Description: RF57-58-61

<b>Feature ID</b>	RF59 (Repository Feature 59)
<b>Name</b>	Export data using XML (METS, MARC)
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	BibUpload, MiscUtil
<b>Description of the new feature</b>	
<p>The repository already offers several formats to export content as XML, both in the main search page and in the detail record page. A new output format to export records is added: METS (Metadata Encoding &amp; Transmission Standard).</p>	
<b>Implementation details</b>	
<p>A new BibFormat element called “bfe_mets” is created. This element retrieves the METS of the corresponding record from the mongoDB and displays it to the user. A new row is added into the “format” table in the database corresponding to the METS output format, where:</p>	




## AstroDAbis

Archived date: 2013/08/22

Original Blog URL: <http://astrodabis.jiscinvolve.org/wp>

### Posts in this blog

#### AstroDAbis 's beta service

 normangray | Posted Date: 2012/03/22 Archived date: 2013/08/22

The AstroDAbis project 's annotation service is now up and running at [astrodabis.roe.ac.uk](http://astrodabis.roe.ac.uk) , for your delight and delectation.

This is a beta service. It works, but we think we still have some work to do to make it a bit more user-friendly, and to integrate it more fully into the set of services that ROE offers. That 's scheduled to happen over the summer, as part of ROE 's intended revamp of its services.

If you spot any problems with the service, please do let us know.

This entry was posted on Thursday, March 22nd, 2012 at 12:33 and is filed under [Uncategorized](#) .

#### Moving gracefully towards the end...

 normangray | Posted Date: 2012/02/17 Archived date: 2013/08/22

The AstroDAbis project is moving towards its final stages. JISC extended our project end-date, to better mesh with an anticipated revamp of associated services within the host unit. That revamp 's been

JPEGs are created by taking snapshots of the records.



## AstroDAbis

Creation Date: 2013-08-22

### Original blog URL

<http://astrodabis.jiscinvolve.org/wp>

<b>AstroDAbis</b> <small>Stand-off annotation for astronomical catalogues</small>	
<b>PAGES</b> <small>About Resources and outputs</small>	<b>AstroDAbis's beta service</b> <small>March 22nd, 2012</small>
<b>ARCHIVES</b> <small>March 2012</small>	<small>The AstroDAbis project's annotation service is now up and running at <a href="http://astrodabis.roe.ac.uk">astrodabis.roe.ac.uk</a>, for your delight and</small>

## Implementation details

- LaTeX TeX Live distribution is used for creating PDF files. “install-texlive” target is added to **Makefile.am** to install required packages.
- PhantomJS is used for taking snapshot of the pages to create JPEG files. “install-phantomjs-64bits” and “install-phantomjs-32bits” targets are added to **Makefile.am** to install required packages.
- **Default\_HTML\_actions.bft** is modified to add PDF and JPEG as an exporting option.
- Four new output formats are introduced:
  - **JPEG.bfo** : Output format to construct the page for taking snapshot.
  - **JPEGB.bfo** : Like **JPEG.bfo** but less detailed.
  - **PDF.bfo** : Output format to construct LaTeX template.
  - **PDFB.bfo** : Like **PDF.bfo** but less detailed.
- The following format templates are introduced to create LaTeX documents and the page to take snapshot for exporting as JPEG:
  - **JPEG.bft**
  - **BlogJPEG.tpl**
  - **CommentJPEG.tpl**
  - **Comment\_JPEG\_brief.tpl**
  - **PostJPEG.tpl**
  - **Post\_JPEG\_brief.tpl**
  - **LaTeX.bft**
  - **BlogLaTeX.tpl**
  - **CommentLaTeX.tpl**
  - **Comment\_LaTeX\_brief.tpl**
  - **PostLaTeX.tpl**
  - **Post\_LaTeX\_brief.tpl**
- The following BibFormat elements are introduced for creating LaTeX documents:
  - **bfe\_latex\_abstract.py**
  - **bfe\_latex\_authors.py**
  - **bfe\_latex\_begin.py**
  - **bfe\_latex\_blog\_posts.py**
  - **bfe\_latex\_blog\_url\_link.py**
  - **bfe\_latex\_comment\_header.py**
  - **bfe\_latex\_default.py**
  - **bfe\_latex\_end.py**
  - **bfe\_latex\_post\_comments.py**
  - **bfe\_latex\_record\_body.py**



- `bfe_latex_record_dates.py`
  - `bfe_latex_snapshot.py`
  - `bfe_latex_tags.py`
  - `bfe_latex_title.py`
- The following BibFormat elements are modified to create the page and take snapshot for exporting as JPEG:
    - `bfe_blog_posts.py`
    - `bfe_post_comments.py`
    - `bfe_record_dates.py`
    - `bfe_snapshot.py`
  - The following templates are created to export as JPEG and PDF.
    - `footer_jpeg.html` : Footer used for exporting as JPEG.
    - `header_jpeg.html` : Header used for exporting as JPEG.
    - `page_jpeg.html` : The base page to inherit format templates for exporting as JPEG.
    - `page_latex.tpl` : The base page to inherit format templates for exporting as PDF.
    - `record_jpeg.html` : Default template to export as JPEG.
  - To get the export requests for PDF and JPEG, `index` endpoint of `websearch_blueprint.py` and `print_records` function in `search_engine.py` is modified.
  - `create_pdf` and `create_jpeg` functions are introduced in `bibformat.py` to create PDF and JPEG files, respectively.
  - `summary` endpoint is introduced in `record_blueprint.py` to construct the page for taking snapshot to export as JPEG.
  - `jpeg_render_script.js` is introduced to use on taking snapshot of the page by PhantomJS.
  - `latexutils.py` and `latexutils_image.py` is created to keep common functions to creating LaTeX document. The following functions are in `latexutils.py` :
    - `begin_document` : Initialization codes for a LaTeX document.
    - `end_document` : Finalization codes for a LaTeX document.
    - `remove_escape_chars` : Returns LaTeX representation of special LaTeX characters.
    - `format_raw_text` : Formats given text as LaTeX.
    - `format_latex_field` : Formats given name and value to display as BibTeX field.
    - `html_to_latex` : Converts HTML to LaTeX document.

The following functions are in `latexutils_image.py` :

- **get\_image\_path** : Returns the path of given image. If the image does not exist in file system, downloads it to an available location.
- **get\_unique\_file\_path** : Returns a unique path to save the image temporarily.
- To convert HTML to LaTeX, **bibformat\_pdf\_with\_latex\_template.py** module is created. It has the following classes:
  - **PdfWithLatexTemplateHtmlParser** class is inherited from **HTMLParser** class.
  - **LatexConverter** : Converts HTML to LaTeX. Possible LaTeX commands are mapped from dictionaries in **bibformat\_pdf\_with\_latex\_config.py** module. Only a few HTML tags are needed extra effort. The methods defined to handle them are called from **PdfWithLatexTemplateHtmlParser** class. It has the following methods:
    - \* **get\_latex\_text** : Returns concatenation of definitions and latex codes.
    - \* **\_handle\_latex\_commands** : Adds given data to one of the buffers holding LaTeX template according to state of HTML tags.
    - \* **handle\_raw\_data** : Main controller for inserting HTML elements' content.
    - \* **\_insert\_raw\_data** : Handles LaTeX special characters, maps special HTML characters (e.g. &hellip) with their LaTeX equivalences and adds the content to latex code buffer.
    - \* **trivial\_tag\_start** : Handles start tags.
    - \* **tag\_end** : Pops end values until popped element's type is given tag. Until an HTML tag is found, CSS related values are added to buffer. For table related elements extra effort is needed.
    - \* **br\_start** : Applies new line only after text.
    - \* **font\_start** : Handles attributes of font tag.
    - \* **anchor\_start** : Handles attributes of anchor tag.
    - \* **img\_start** : Handles img tag based on parameters in config module. If **CFG\_BIBFORMAT\_LATEX\_USE\_LOCAL\_IMAGES** is not set, then the image is downloaded from src attribute value. If the image extension is not supported by LaTeX, converts image into specified format.
    - \* **table\_start** : Handles start tag of table.
    - \* **table\_end** : Handles end tag of table.
    - \* **table\_row\_start** : Handles start tag of tr.
    - \* **table\_row\_end** : Handles end tag of tr.
    - \* **table\_cell\_start** : Handles start tag of td or th.
    - \* **flush\_end\_stack** : If there is not any problem with the HTML content, all HTML tags are successfully matched. If there are any element for style rules, pops them and adds 'end' values to LaTeX text.

- \* **add\_style** : Adds style related LaTeX codes based on current tag's CSS rules. Most of the CSS declarations are mapped from dictionary. However, some needs extra effort (e.g. font-size, color, border, etc.)
- \* **\_\_search\_style\_in\_external\_css** : In all possible selectors' CSS rules, searches given property.
- \* **extract\_style** : Finds all possible CSS declarations.
- \* **find\_style\_selectors** : Checks current HTML element's parents and their ids, class names to construct CSS selectors possibly containing more than one identifier (e.g. ".class\_name >#id", "div div img", "div.class\_name >p", etc.)
- **CssParser** : Parses the CSS rules and converts into dictionary. It has the following methods:
  - \* **parse** : Finds each CSS declaration and selectors of CSS rules.
  - \* **parse\_inline\_style** : Rather than whole css file data, runs on only inline CSS.
  - \* **\_\_extract\_css\_declarations** : Converts style rule declarations in str form to dictionary.
  - \* **\_\_read\_css\_files** : Reads CSS file. This method is called when the data is supplied as file names.
- Configurations for converting HTML to LaTeX take place in **bibformat\_pdf\_with\_latex\_template\_config.py** . It has the following configurations:
  - **CFG\_BIBFORMAT\_EXPORT\_DIR** : The directory to keep files of exported record.
  - **CFG\_BIBFORMAT\_LATEX\_TEMP\_DIR** : The directory to keep temporary files when creation PDF from LaTeX.
  - **CFG\_BIBFORMAT\_PATH\_PDF\_CONVERTER** : The path of *xelatex* tool.
  - **CFG\_BIBFORMAT\_LATEX\_KEEP\_FILE\_EXTENSIONS** : The file extensions which will be kept in **CFG\_BIBFORMAT\_EXPORT\_DIR** after *pdflatex* ends its job.
  - **CFG\_BIBFORMAT\_CSS\_FILES** : The paths of the CSS files used when converting HTML to LaTeX template.
  - **CFG\_BIBFORMAT\_IMG\_SRC\_URL** : Regular expression to check *src* attribute of *img* whether it is a local path or a URL.
  - **CFG\_BIBFORMAT\_LATEX\_HEADER** : The first line of the LaTeX document.
  - **CFG\_BIBFORMAT\_LATEX\_PAGE\_SETTINGS** : Options for page layout.
  - **CFG\_BIBFORMAT\_LATEX\_BEGIN\_DOC** : The *begin* tag of the LaTeX document.
  - **CFG\_BIBFORMAT\_LATEX\_END\_DOC** : The *end* tag of the LaTeX document.

- **CFG\_BIBFORMAT\_LATEX\_PACKAGES** : The packages that are used in LaTeX document.
- **CFG\_BIBFORMAT\_LATEX\_SPECIAL\_CHARS\_REGEX** : Regular expression to identify special characters in LaTeX.
- **CFG\_BIBFORMAT\_LATEX\_SPECIAL\_CHARS\_LIST** : The list of special characters in LaTeX.
- **CFG\_BIBFORMAT\_LATEX\_MATH\_FORMULAS** : Regular expression to find patterns of math formulas which are in already LaTeX format.
- **CFG\_BIBFORMAT\_LATEX\_ONLY\_TEXT** : Regular expression to split content into pieces according to start and end of math formulas.
- **CFG\_BIBFORMAT\_LATEX\_LONG\_WORDS** : Regular expression to find words containing symbols more than provided number.
- **CFG\_BIBFORMAT\_MATHJAX\_ENABLED** : Set to 1 to seek LaTeX formulas in text. If it is set to 0, then **CFG\_BIBFORMAT\_MATHJAX\_DELIMITERS** is ignored.
- **CFG\_BIBFORMAT\_MATHJAX\_DELIMITERS** : Delimiter characters for formulas.
- **CFG\_BIBFORMAT\_CSS\_PARSER\_REGULAR\_EXPRESSIONS** : Regular expressions used for parsing CSS.
- **CFG\_BIBFORMAT\_CSS\_SELECTOR\_COMPONENTS** : Regular expression to identify the components of a CSS selector.
- **CFG\_BIBFORMAT\_LATEX\_CSS\_SELECTORS** : Complex selector types for which CSS rules will be applied.
- **CFG\_BIBFORMAT\_LATEX\_STYLE\_FIRST\_TAGS** : The list of the tags to apply CSS rules before handling.
- **CFG\_BIBFORMAT\_LATEX\_NON\_END\_TAGS** : The tags that may not have '/' character indicating end of tag.
- **CFG\_BIBFORMAT\_LATEX\_EXCEPTIONAL\_COMMANDS** : List of commands after which an environment can not be initialized.
- **CFG\_BIBFORMAT\_LATEX\_DEFAULT\_IMG\_FORMAT** : Default image format for LaTeX.
- **CFG\_BIBFORMAT\_LATEX\_SUPPORTED\_IMG\_EXTS** : The list of supported image types.
- **CFG\_BIBFORMAT\_LATEX\_PX\_CM\_SCALE** : Cm value that corresponds to 1 pixel.
- **CFG\_BIBFORMAT\_LATEX\_A4\_SCALE** : Amount of scale for A4 paper dimensions.
- **CFG\_BIBFORMAT\_LATEX\_DEFAULT\_IMG\_SCALE** : The rate to scale the image.

<ul style="list-style-type: none"> <li>– <b>CFG_BIBFORMAT_LATEX_DEFAULT_FONT_SIZE</b> : Default font size for LaTeX document.</li> <li>– <b>CFG_BIBFORMAT_LATEX_KB_FILE</b> : The path of the latex-to-unicode.kb file.</li> <li>– <b>CFG_BIBFORMAT_LATEX_SPECIAL_CHARS_EQUIVALANCES</b> : Representations of special LaTeX characters.</li> <li>– <b>CFG_BIBFORMAT_LATEX_COMMAND_CONSTRAINTS</b> : Constraints for LaTeX tags.</li> <li>– <b>CFG_BIBFORMAT_LATEX_ALIGNMENTS</b> : LaTeX representations of HTML alignment options.</li> <li>– <b>CFG_BIBFORMAT_LATEX_COMMANDS</b> : LaTeX commands used for rather than tag and CSS mappings.</li> <li>– <b>CFG_BIBFORMAT_LATEX_REPRESENTATION_OF_HTML_TAGS</b> : HTML tags and their LaTeX representations.</li> <li>– <b>CFG_BIBFORMAT_LATEX_REPRESENTATION_OF_CSS_RULES</b> : CSS declarations and their LaTeX representations for possible values.</li> <li>– <b>CFG_BIBFORMAT_HTML_SPECIAL_CHARS_BY_NUM</b> : Unicode numbers of special HTML chars and their LaTeX representations.</li> <li>– <b>CFG_BIBFORMAT_HTML_SPECIAL_CHARS_BY_NAME</b> : Special HTML chars and their LaTeX representations.</li> </ul>		
<table border="1"> <tr> <td><b>Implemented By</b></td> <td>Senan Postaci(SRDC)</td> </tr> </table>	<b>Implemented By</b>	Senan Postaci(SRDC)
<b>Implemented By</b>	Senan Postaci(SRDC)	

Table A.34: Implementation Description: RF62

<b>Feature ID</b>	RF64 (Repository Feature 64)
<b>Name</b>	The archive offers the option to login using external (universal) credentials
<b>Effort Spent</b>	3 Weeks
<b>Modules Affected / Created</b>	MiscUtil, WebAccess, WebSession

#### Description of the new feature

This feature was implemented by BlogForever developers, but ported to next and integrated into Invenio core by Invenio developers. Therefore, now it is a

part of Invenio and directly used in BlogForever repository. Description of the functionality of this feature can be found in D4.5.

<b>Implemented By</b>	SRDC
-----------------------	------

Table A.35: Implementation Description: RF64

<b>Feature ID</b>	RF65 (Repository Feature 65)
<b>Name</b>	The archive analyzes blog links and stores the connections between them separately
<b>Effort Spent</b>	4 weeks
<b>Modules Affected / Created</b>	-

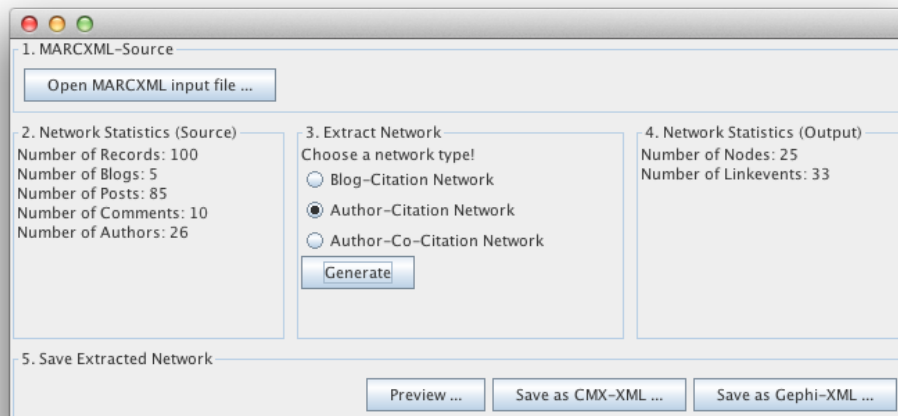
#### Description of the new feature

A network consists of nodes and links between the nodes. Three kinds of networks are extracted from the archived blogosphere:

- Blog-Citation network
  - Nodes: All Blog objects in the archived blogosphere.
  - Links: A link is created when a hyperlink in a post references another post or a blog. The link goes from the parent blog of the sending post to the parent blog of the receiving post or the receiving blog.
- Author-Citation network
  - Nodes: All authors in the archived blogosphere.
  - Links: A link is created when a hyperlink in a post or in a comment references another post or comment. Sending and receiving posts or comments must have an author.
- Author-Co-Citation network
  - Nodes: All author objects in the archived blogosphere.
  - Links: A link is created if two or more posts or comments reference the same address through a hyperlink.

The Blog-Citation network and the Author-Citation network consider only links that direct to an object that is archived in the repository. The Author-Co-Citation network considers all kind of links (external and internal).

The implementation of the feature differs from the feature design: Links are not differentiated between Citations, BlogRoll, and Pingback/Trackback because these information can not be identified by the spider.



RF65 - GUI of the feature. The main steps in the network generating process are (a) to choose the input, (b) to choose the network type, and (c) to choose the output format and location

### Implementation details

The feature is implemented as java application together with the repository feature RF72 which visualizes the extracted network. It has been also ported to Python and this is the version that has been integrated into the repository.

Nodes are represented as objects that have an ID and a maximum of five details. Links are an aggregation of one or more Linkevents. A linkevent has one sender, one or more recipients, a timestamp, and a maximum of five details. The distinction of Linkevents with timestamps allows representing and analysing the network evolution over time.

The application accepts as input a collection of blog, post, and comment objects in MARCXML from the repository. The objects in the collection are analyzed and the network extracted by the Java class *tub.BFMarcXMLReader* and stored in the Java objects

- *tub.dataElements.Network*
- *tub.dataElements.Node*
- *tub.dataElements.Linkevent*

Currently, the two output formats CMX-XML and GEXF are provided.

- With CMX-XML, the extracted network can be further analysed and explored with the Commetrix<sup>a</sup> software. The Java class *tub.CMXMLWriter* transforms the extracted network into CMX-XML. The CMX-XML preserves the structure of link events, and, therefore, Commetrix facilitates dynamic and static analysis.
- GEFX can be analysed with Gephi<sup>b</sup>, an open-source tool to visualize and analyse networks. The Java class *tub.GEXFWriter* transforms the extracted network into GEXF. Thereby, the linkevents has to be aggregated to edges due to the limitations of the GEXF format. Therefore, the Gephi tool facilitates only static analysis.

For each of these two output formats a BibFormat template has been created: *format\_records\_cmx.tpl* and *format\_records\_gexf.tpl*

The Java class *tub.BFNetworkGenerate* contains the GUI for the Network Generator application.

<sup>a</sup><http://www.commetrix.de/>

<sup>b</sup><https://gephi.org/>

<b>Implemented By</b>	TUB
-----------------------	-----

Table A.36: Implementation Description: RF65

<b>Feature ID</b>	RF66 (Repository Feature 66)
<b>Name</b>	The archive provides a historical/chronological navigation
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	WebBlog, BibFormat

#### Description of the new feature

A new BibFormat element has been created and included in the BlogHTML template. This element displays a JavaScript slider showing the blog posts in a slider timeline. The events (post published) can be clicked and a short description of the record is displayed in a bubble, with a link to the detailed view of the record.

The slider is populated with data with an XML file created with the information of the blog as a hidden XML export option. It could easily be extended to include also in the XML file the comments of each post thus enriching the content of the timeline slider.





### Implementation details

- Added a new install option to the Makefile options: ( **install-jstimeline-plugin** ) that installs the necessary JavaScript files. The code used is from the **simile-timeline.2.3.0** plugin.
- Added XML output format to feed the plugin. The format is called **xtl** and can be viewed adding **xtl** to the url in the detailed view of a blog.
- Added a BibFormat element called **bfe.blog.timeline.py** to be used in **BlogHTML.bft**

Implemented By	CERN
----------------	------

Table A.37: Implementation Description: RF66

<b>Feature ID</b>	RF67 (Repository Feature 67)
<b>Name</b>	The archive fetches and stores embedded content
<b>Effort Spent</b>	3 Weeks
<b>Modules Affected / Created</b>	BibUpload, WebSchedule

### Description of the new feature

The repository fetches the embedded content using the spider's API. Afterwards, it uses BibUpload to insert the metadata and the embedded files into the repository databases, as described in RF9.

### Implementation details

The script `spider_repository_communication.py` establishes a connection with the spider, retrieves the list of new records, and for each one of them downloads the files (metadata and embedded content) and calls the BibUpload module that inserts them into the repository.

<b>Implemented By</b>	Raquel Jiménez, Jaime García (CERN)
-----------------------	-------------------------------------

Table A.38: Implementation Description: RF67

<b>Feature ID</b>	RF70 (Repository Feature 70)
<b>Name</b>	6 weeks
<b>Effort Spent</b>	WebAccess, WebSession
<b>Modules Affected / Created</b>	The archive can provide services under some cost using a billing system

### Description of the new feature

This feature introduces “premium access to collections” service. With this feature admin can restrict collections for some cost for a finite/infinite time interval.

Premium packages can be managed easily from the admin panel.  
(Administration ⇒ Configure Webaccess ⇒ Manage Premium Packages)

Admin may add premium packages for collections:

Current premium packages can be monitored, edited or deleted:

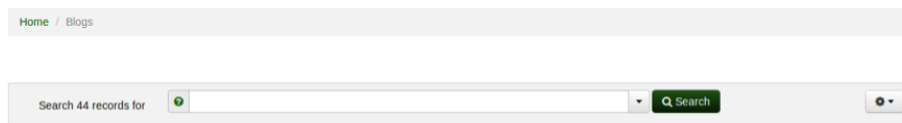
[Edit Premium Packages](#)
[Payment History](#)
[Premium Users](#)

ID	Name	Details	Duration	Price	Access			
1	Blogs and Comments	Display blogs and comments for 3 days!	3 Day	5.0 EUR	Blogs Comments			
2	Posts - 1	Displays posts for a month!	1 Month	19.0 EUR	Posts			
3	Pages	Pages for a year!	1 Year	49.99 EUR	Pages			
4	All in One	All in One unlimited membership!!!	Unlimited	149.99 EUR	Blogs Posts Comments Pages			

[Add new premium package](#)

- Edits the premium package (Displays same form as adding new one)
- Deletes the premium package
- Moves the premium package up and down respectively. The order of the premium packages can be changed through these buttons.

When premium packages are added for certain collections, these collections become restricted.



This collection is restricted. If you are authorized to access it, please click on the Search button.

After edit or delete operations, if a collection loses its premium packages, that collection becomes unrestricted.

By clicking search button, if the user has not bought a premium package, the list of the premium packages related to that collection are shown.

### Purchase a premium package

Name	Details	Duration	Price	Access to collections
<input type="radio"/> Blogs and Comments	Display blogs and comments for 3 days!	3 Days	5.0 EUR	Blogs Comments
<input type="radio"/> All in One	All in One unlimited membership!!!	Unlimited	149.99 EUR	Blogs Posts Comments Pages



To purchase a premium package, users can select a suitable package and payment method which are the following:

#### 1) Credit card

## Purchase a premium package with credit card

### Premium Package Details

<b>Name</b>	Blogs and Comments
<b>Details</b>	Display blogs and comments for 3 days!
<b>Duration</b>	3 Days
<b>Price</b>	5.0 EUR
<b>Access to Collections</b>	Blogs Comments

### Credit Card Information

Name on Card	<input type="text"/>
Card Number	<input type="text"/>
Expiration	<input type="text" value="1"/>   <input type="text" value="2013"/>
Security Code	<input type="text"/>
First Name	<input type="text"/>
Last Name	<input type="text"/>
Street	<input type="text"/>
City	<input type="text"/>
State / Province	<input type="text"/>
ZIP / Postal Code	<input type="text"/>
Country	AALAND ISLANDS

[Upgrade](#)

The user can buy premium packages with his/her credit card through the above form. This is the last screen before completing the transaction. After clicking the upgrade button, if the transaction fails, an error message is shown:

### Credit Card Information

This transaction cannot be processed. Please enter a valid credit card number and type.

Otherwise, a confirmation page is displayed.

You have successfully purchased the premium package. Please note that your transaction ID is Paypal-5HG92527WD4206728

## Your Premium Memberships

### Your current premium membership status:

Collection	Status	
Blogs	Your membership will be expired in 2013-05-23 18:00:24	<a href="#">Extend your membership!</a>
Posts	You don't have any premium membership to display!	<a href="#">Buy a premium package</a>
Comments	Your membership will be expired in 2013-05-23 18:00:24	<a href="#">Extend your membership!</a>
Pages	You don't have any premium membership to display!	<a href="#">Buy a premium package</a>

## 2) PayPal Express Checkout

**BlogForever Repository Demo**

**Your order summary**

Descriptions	Amount
Posts - 1 Item description: Display posts for ... Item price: €19.00 Quantity: 1	€19.00
<b>Item total</b>	<b>€19.00</b>
<b>Total €19.00 EUR</b>	

**Review your information**

**PayPal**

[Continue](#)

**Payment methods** [Change](#)

PayPal Balance \$26.60 USD

PayPal Conversion Rate as of May 22, 2013: 1 U.S. Dollar = 0.714485 Euros

PayPal gift card, certificate, reward, or other discount [Redeem](#)  
View [PayPal policies](#) and your payment method rights.

---

**Contact information**  
alper\_1347461937\_per@srdc.com.tr

[Continue](#)

You're almost done. You will confirm your payment on BlogForever Repository Demo.

[Cancel and return to BlogForever Repository Demo.](#)

User may choose PayPal express checkout if s/he has a PayPal account. Clicking “Checkout with PayPal” button redirects the user to the PayPal page to login and confirm the transaction. After clicking “Continue Button”, user is redirected back to Invenio site, and confirms his/her order.

Home / Your Account / Your Premium Memberships / Confirm payment

## Confirm payment

Name	Details	Duration	Price
Posts - 1	Display posts for a month!	1 Month	19.0 EUR
Total: 19.0 EUR			

User clicks the checkout with PayPal button and confirms the transaction.

You have successfully purchased the premium package. Please note that your transaction ID is Paypal-7JX13215RN579620K x

## Your Premium Memberships

Your current premium membership status:

Collection	Status
Blogs	You don't have any premium membership to display! <a href="#">Buy a premium package</a>
Posts	Your membership will be expired in 2013-06-21 15:52:49 <a href="#">Extend your membership!</a>
Comments	You don't have any premium membership to display! <a href="#">Buy a premium package</a>
Pages	You don't have any premium membership to display! <a href="#">Buy a premium package</a>

Users may see their premium group memberships via “Your account page”:

## Your Account

Filter widgets

**Account** ↗

You are logged in as Demo

**Premium Memberships**

You are able to display 1 of total 4 premium collections.

**Baskets WIP**

You have 0 personal baskets

**Alerts WIP**

You own the following 0 alerts

**Searches** ↗

You have made 0 queries. A detailed list is available with a possibility to (a) view search results and (b) subscribe to an automatic email alerting service for these queries.

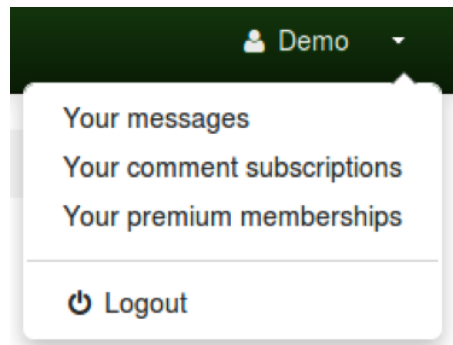
**Messages** ↗

You have 0 new messages out of 0 messages.

**Group** ↗

You are not involved in any group.

Users may display its detailed status of premium memberships by clicking the title of the “Premium Memberships” box or “Your premium packages” link on the top of the page.



“Your Premium Memberships” page consists of two main parts: “Premium membership status” and “Available premium packages”

### Your Premium Memberships

Your current premium membership status:

Collection	Status
Blogs	Your membership will be expired in 2013-05-25 16:02:07 <a href="#">Extend your membership!</a>
Posts	Your membership will be expired in 2013-06-21 15:52:49 <a href="#">Extend your membership!</a>
Comments	Your membership will be expired in 2013-05-25 16:02:07 <a href="#">Extend your membership!</a>
Pages	You don't have any premium membership to display! <a href="#">Buy a premium package</a>

Available premium packages:

Name	Details	Duration	Price	Access to collections
<input type="radio"/> Blogs and Comments	Display blogs and comments for 3 days!	3 Days	5.0 EUR	Blogs Comments
<input type="radio"/> Posts - 1	Display posts for a month!	1 Month	19.0 EUR	Posts
<input type="radio"/> Pages	Pages for a year!	1 Year	49.99 EUR	Pages
<input type="radio"/> All in One	All in One unlimited membership!!!	Unlimited	149.99 EUR	Blogs Posts Comments Pages



On upper part, the user may display the expiration date of his/her premium memberships, extend it by the “Extend your membership!” or “Buy a premium package” links.

On lower part, user may see all of the available packages and purchase one of them.

The admin may see the transaction history and premium members from the admin panel:

**Premium Area** selection for WebAccess Admin

Edit Premium Packages Payment History Premium Users

Number of transactions : 2

Total revenue : 24.00 EUR

Transaction Time	User ID	User e-mail	Package ID	Price	Payment Method
2013-05-22 16:02:07	2	jekyll@cds.cern.ch	1	5.0 EUR	Credit Card
2013-05-22 15:52:49	2	jekyll@cds.cern.ch	2	19.0 EUR	Paypal

**Premium Area** selection for WebAccess Admin

Edit Premium Packages Payment History Premium Users

Total number of premium users: 1

User ID	User E-mail	User Nickname	Collection	Expiration Time
2	jekyll@cds.cern.ch	Demo	Blogs	2013-05-25 16:02:07
2	jekyll@cds.cern.ch	Demo	Comments	2013-05-25 16:02:07
2	jekyll@cds.cern.ch	Demo	Posts	2013-06-21 15:52:49

Give a gift!

The admin can disable paying with credit card, paypal express checkout or whole premium service from **invenio.conf** file. Credit card payment can be done by PayPal or Ogone payment gateways. Admin can select payment methods to use in **invenio.conf** . Also the API credentials of PayPal and Ogone can be set in this file.

If the admin disables the premium service, users cannot buy premium packages but restriction status of collections does not change. If the admin enables it back, users can buy premium packages again.

## Implementation details

- **12 new variables added into invenio.conf :**
  - **CFG\_PREMIUM\_SERVICE** : **1** for enabling, **0** for disabling the premium service. If this variable is changed **1** to **0** , users cannot buy premium packages but restriction status of collections does not change. On the contrary, If this variable is changed from **0** to **1** , users can buy premium packages again.

- **CFG\_TEST\_PREMIUM\_SERVICE** : **1** for using test servers of the payment gateways, otherwise **0** .
  - **CFG\_CREDIT\_CARD\_PAYMENT\_GATEWAY** : The payment gateway used for purchasing with credit card. It may have 3 options: **“paypal”** , **“ogone”** or **“”** (blank). If it is blank, paying with credit card becomes disabled.
  - **CFG\_USE\_PAYPAL\_EXPRESS\_CHECKOUT** : The variable to decide to use “Paypal Express Checkout” or not. **1** for enabling, **0** for disabling.
  - **CFG\_PAYPAL\_API\_USERNAME** , **CFG\_PAYPAL\_API\_PASSWORD** , **CFG\_PAYPAL\_API\_SIGNATURE** , **CFG\_PAYPAL\_API\_VERSION** : The credentials to use PayPal API.
  - **CFG\_OGONE\_API\_PSPID** , **CFG\_OGONE\_API\_USERID** , **CFG\_OGONE\_API\_PSWD** : The credentials to use OGone API.
  - **CFG\_PREMIUM\_GROUP\_SUFFIX** : The suffix of the premium group names.
- **4 new tables added into database:**
    - **premium** : Keeps the information (name, details, duration, price and display order) about premium packages.
    - **hstPAYMENT** : Keeps the payment history.
    - **premium\_collection** : keeps the premium package - collection mapping.
    - **collection\_accrole** : keeps the collection - role mapping
  - **If a collection needs a premium membership to be accessed, it is checked when displaying corresponding collection**

**search** function in **websearch\_blueprint.py** is modified to accommodate billing system. It checks if the premium service is enabled and if there is a premium package to access that collection. If a user does not have a right to access that collection, the list of the premium packages that allows to access is displayed.
  - **The admin panel to manage premium packages is introduced**

With the admin panel, premium packages can be added, edited and deleted. In addition, their display order may be changed. Payment history can be monitored and the list of the users who is a member of the premium groups can be displayed in the admin page. To achieve these functionalities, necessary functions are added into **webaccess\_admin\_blueprint.py** and **webaccess\_admin\_premiumarea.html** is added to templates.
  - **WebPayment module is introduced** It handles the cases about premium service, contains necessary functions and classes. **webpayment.py** contains the following functions:



- **add\_new\_premium\_package** : Adds a premium package to the database. Arranges the restrictions for collections of the new premium package.
- **edit\_premium\_package** : Edits the premium package. Arranges the restrictions for collections of the premium package.
- **fix\_roles\_and\_authorizations** : Arranges the restrictions for the premium collections.
- **add\_role\_and\_authorization** : Restricts the given collection. Adds a role and an authorization to restrict the collection.
- **grant\_user\_access** : Give a user the access to the collections restricted a given premium package.
- **gift\_premium\_package** : Gives a premium package to a user as gift.
- **get\_possible\_packages** : Returns the premium packages to display given collection.
- **get\_package\_collection\_map** : Returns a dictionary that contains information about which premium package allows to access which collections.
- **register\_payment\_history** : Registers the transaction to payment history.
- **delete\_premium\_package** : Deletes the premium package with given ID. Arranges the restrictions of the collections that can be displayed by this premium package.
- **get\_user\_premium\_membership** : Returns total number of premium packages that the user has purchased.
- **get\_membership\_expiration\_dates** : Returns the expiration dates of the premium memberships of the user with given ID.

WebPayment module provides **webpayment\_query.py** file which contains database related functions of WebPayment module. It also provides **webpayment\_base.py** which includes the following required classes to implement payment gateways:

- **PaymentGatewayResponse** : This class is inherited from dict class. It ensures that the dictionary contains corresponding premium package, success state of the transaction, transaction ID, error messages and additional data if exists.
- **PaymentGateway** : This is an “abstract” class from which all of the payment gateways should be inherited. The fields which should be **overridden** are following:
  - \* **SERVER** :The URL of the payment gateway API.
  - \* **TEST\_SERVER** :The test URL of the payment gateway.
  - \* **\_additional\_inputs** :Some payment gateways require more data than the credit card information. This field is used to specify the additional information required to payment gateway.
  - \* **name** :The name of the payment gateway.

- \* `_accept_types` :List of the credit card types that accepted. Types can be `PaymentGateway . VISA` , `PaymentGateway . MASTERCARD` , `PaymentGateway . DISCOVER` , `PaymentGateway . AMERICANEXPRESS` or `PaymentGateway . MAESTRO` constants.

If the payment gateway is used for purchasing with credit card, `process` method should be **overridden**. `process` performs the credit card transaction and return the response ( `PaymentGatewayResponse` ) including transaction ID if succeeded, error messages if failed.

If the payment gateway redirects the user to a 3rd party site to complete the payment, `construct_checkout_url` , `get_transaction_details` and `complete_transaction` methods should be overridden:

- \* `construct_checkout_url` :This method should return the response with 3rd party site URL to checkout in the `data` field of the response. If fails, it should return response with error messages. The return URL when calling the payment gateway api should be in the form of `CFG_SITE_SECURE_URL /webpayment/review?id_package=self.premium_package.id&payment_method=self.name`. If you want to show the user what s/he is buying, the endpoint should be `review` and override `get_transaction_details` , or you may complete the payment after returning from 3rd party site by setting endpoint as `complete`.
- \* `get_transaction_details` : Checks if the transaction is appropriate for the payment gateway. If it is, it returns the HTML code of the button for completing the transaction in `data` key of the response. Otherwise, it should return a response with error messages. If you want to skip this step, just do not override this function.
- \* `complete_transaction` : Should complete the transaction. If the transaction is succeeded, it should return a response with transaction id. Otherwise, it should return a response with error messages.

`webpayment_paypal.py` and `webpayment_ogone.py` modules contain classes derived from `PaymentGateway` . These two both contain necessary methods to buying with credit card. In addition, `webpayment_paypal.py` contains methods for completing payment in 3rd party site (PayPal Express Checkout).

- Payment methods can be managed from `webpayment_config.py` and implemented methods should be added into this file.

**CFG\_CREDIT\_CARD\_PAYMENT\_METHODS** is a dictionary which contains the classes implemented for purchasing with credit card. Its keys are the name of the payment methods (with uppercase letters) and values are only the corresponding classes (not instances).

**CFG\_PAYMENT\_METHODS** is also a dictionary which contains all the payment methods for purchasing with 3rd party site (like PayPal Express Checkout). Its values are the name of the payment methods and values are the corresponding class. In addition it has the key **cc** for credit card payment whose value is determined by **invenio.conf** .

- **New modules are added for the web interface**

- **webpayment\_user\_settings.py** is introduced to add “Your premium memberships” box to “Your account” page.
- **webpayment\_blueprint.py** and templates **webpayment\_display.html** , **webpayment\_index.html** , **webpayment\_packages.html** , **webpayment\_review.html** , **webpayment\_upgrade.html** are introduced.

Implemented By	SRDC
----------------	------

Table A.39: Implementation Description: RF70

<b>Feature ID</b>	RF71 (Repository Feature 71)
<b>Name</b>	The archive provides a personalized annotating and highlighting tool for users
<b>Effort Spent</b>	4 weeks
<b>Modules Affected / Created</b>	WebSearch, WebSession, Webstyle, Miscutil, Bibformat

#### Description of the new feature

This feature enables users to highlight certain parts of the records and add notes on them. It can be enabled/disabled for each collection, So an icon is added to activate the feature at the right of the page. This icon is only visible when a user logged in, otherwise it is not displayed.

## Clean Dissertations?

that will propel me through the end of the process and one that I  
nd, here is the beginning of some thoughts ...

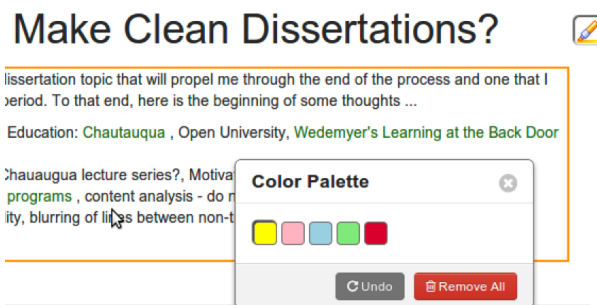
Chautauqua , Open University, Wedemyer's Learning at the Back Door

When the icon is clicked, a color palette containing highlight colors appears and previously saved highlighted items are loaded. 5 colors are chosen as default. These can be changed from [invenio.conf](#) . Palette has also two more options, **remove all** and **undo**.

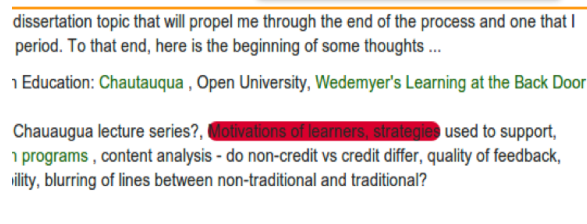
Actions which can be undone by **undo** operation:

- Creating Highlight
- Extending Highlight
- Deleting Highlight
- Adding Note
- Editing Note
- Deleting Note
- Remove All

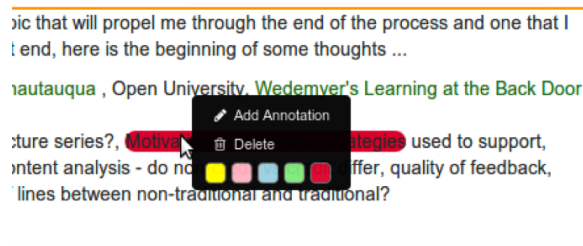
After activating highlight, an orange box appears around the record, which defines the borders of the editable area.



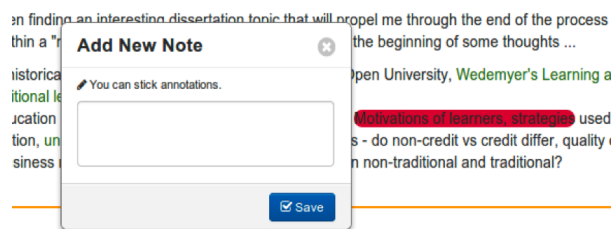
The user is able to highlight the record by holding mouse button and dragging the cursor.



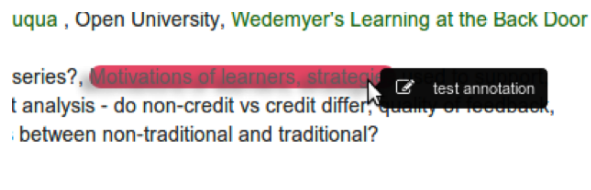
When mouse is rolled over an highlighted area, an **edit menu** is displayed. It provides of 3 options which are **Add Annotation**, **Delete highlight** and **Change Color**.



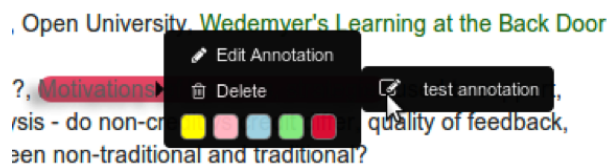
A box is displayed when user clicks on **Add Annotation** so that, the user can enter his/her annotation and save it to the repository.



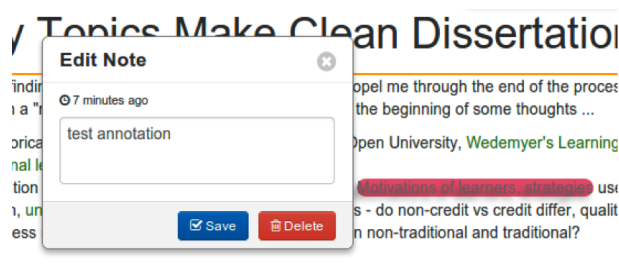
After the note is saved, the text of the corresponding highlight becomes shadowed. When you hover the mouse through annotated section, a tooltip showing the note appears.



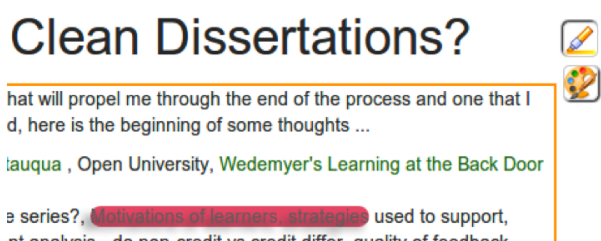
When the icon at the leftmost of the annotation tooltip is clicked, a menu similar to the edit menu appears. It provides of 3 options namely **Edit Annotation**, **Delete Highlight** and **Change Color**.



When a highlighted text that has annotation or **Edit Annotation** link illustrated above is clicked, the corresponding note can be seen. It can be edited and saved or removed completely.



When the color palette is closed, a new icon to reopen the color palette appears just below the highlight activation icon.



Highlighted texts can be extended. If the selected text contains any highlighted part and if the selection color is same with its color, they are merged. Moreover, two neighbor highlighted parts are merged, if they have same color after a color change is applied.

Since there are different nodes in HTML based records, the user selection is resulted in divided highlighted parts. When the user adds an annotation to one of them, it is also added to all highlight nodes with the same identifier. Highlighted parts still seem divided, but they are logically unified.

If selection contains a part of a MathJax expression, it highlights whole MathJax element in order not to ruin structure of MathJax.

## Implementation details

- This feature can be enabled for specific collections by adding collection name to **CFG\_HIGHLIGHT\_AND\_ANNOTATION\_COLLECTIONS**. **Posts** and **Comments** are in this list by default.
- To enable highlighting feature for any content, it is just enough to surround the content with `<div class='highlightable'></div>` tags.
- **selectionRange** object is mainly used for highlight. Therefore, this feature is applicable on Internet Explorer from version 9.
- To highlight the selected text, an element with tag **highlight** is inserted around the selection.
- A **highlight** element does not contain any other **highlight** elements.

- The main aim is keeping the **highlight** elements with minimum depth in DOM tree. To do so, after each selection, the recently inserted **highlight** nodes are traversed. If all its siblings are **highlight** nodes, then just highlight the parent instead of all children. This reduces the costs to save highlights.
- For each text selection, the resulted **highlight** nodes are given same identifier to logically unify them. This identifier is unique for each selection. The identifiers are also used as annotation ID's.
- Highlights are saved as serialized JSON string. On page load, the DOM tree is reconstructed with this JSON object. An example json string:

```
{
  "leaves":{
    "43":[{"s": 22, "e": 35,"a": 0, "id": "2",
           "n": 0, "c": "rgb(255, 255, 0)"}],
    "73":[{"s": 357, "e": 406, "a": 1, "id": "0",
           "n": 0, "c": "rgb(255, 255, 0)"}]
  },
  "nodes":{
    "57":{"c": "rgb(255, 255, 0)", "id": "1", "a": 0}
  }
}
```

'leaves' key corresponds to highlight elements around text nodes. It has keys as numbers which are ids of each dom element. These ids are assigned when highlight mode is on. For the highlight nodes around text nodes, their parent nodes are used as keys i.e 43, 73. Highlight nodes under them are listed in dictionary format.

- 's' denotes start position
- 'e' denotes end position
- 'a' denotes whether that highlight element has an annotation. if yes **1** , otherwise **0** values are used.
- 'id' denotes id '**high\_anno\_id**' attr value of the **highlight** nodes. It is used to keep track of seperated highlight nodes which are result of a selection. For example, the user makes a selection starting from a **<p>** element and ending to another **<div>** . There are more than one highlight nodes as a result of this selection. This value is also used for annotation id.
- 'n' denotes the child number that the indices are valid on.
- 'c' denotes the color.

'nodes' key corresponds to a node level highlighting information. For nodes, there is no need to save indices, it is enough to keep color information, since each node will have a unique identifier (57 in example above) and they will be highlighted directly. The 'c' , 'id' , and 'a' keys have same meanings as above.

<ul style="list-style-type: none"> <li>• Each change is directly saved into the database.</li> <li>• Before loading highlights, record's last modification date and highlight date are compared to understand whether the record is changed or not. If the comparison indicates a change on the record, the user is warned about possible distortion on highlights.</li> <li>• Two new tables namely <b>bibrec_highlights</b> and <b>bibrec_annotations</b> have been inserted into database to keep highlights and annotations.</li> <li>• In <b>websession_webinterface.py</b> , <b>savehighlights</b> , <b>loadhighlights</b> , <b>saveannotation</b> , <b>getannotation</b> and <b>removeannotation</b> methods have been added for communication between server and client sides.</li> <li>• In <b>webuser.py</b> , <b>check_bibrec_modification_date</b> , <b>set_user_bibrec_annotation</b> , <b>get_user_bibrec_annotation</b> , <b>delete_user_bibrec_annotation</b> , <b>set_user_bibrec_highlights</b> , <b>get_user_bibrec_highlights</b> functions have been implemented for database transactions.</li> <li>• In <b>websession_templates.py</b> , <b>tmpl_highlight_tools</b> , <b>tmpl_annotation_box</b> , <b>tmpl_color_palette</b> methods have been implemented to create HTML codes for highlight tools such as color palette and annotation box.</li> <li>• In <b>dateutils.py</b> , <b>difference_between_times</b> function has been inserted to calculate elapsed time in units such as second, minute, hour etc.</li> <li>• <b>CFG_COLOR_PALETTE</b> parameter has been added to define highlight colors. As default, four colors have been set.</li> <li>• Also in <b>inveniocfg.py</b> and <b>search_engine.py</b> , some minor modifications have been made.</li> </ul>	SRDC
---	------

Table A.40: Implementation Description: RF71

<b>Feature ID</b>	RF72 (Repository Feature 72)
<b>Name</b>	The archive provides a visualization of the blogs network structure
<b>Effort Spent</b>	4 weeks
<b>Modules Affected / Created</b>	-
<b>Description of the new feature</b>	
<p>The exploration of a blog network needs a visual representation of the nodes and links. The feature provides a visualization with several options. Thus, a</p>	

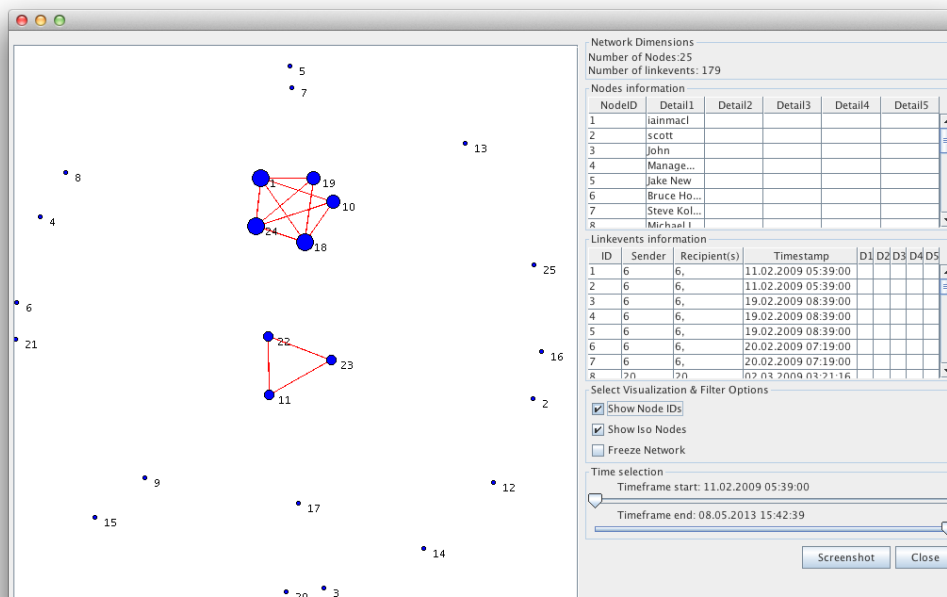


first exploration of the network can be conducted and the network adapted before it is further analysed with more sophisticated analysing software, e.g. Commetrix.

In the example figure below, an Author-Co-Citation network is shown. The network is visualized as blue nodes with red edges between them. The size of a node represents how many linkevent the node sends to another node. This means how often an author references something that somebody has referenced before. The nodes and linkevents of the feature are listed in the corresponding tables on the right side for a more detailed inspection. The visualization has the following options:

- Show Node IDs: Turns on/off that the node ID is shown as label for a node.
- Show Iso Nodes: Turns on/off that isolated nodes are shown. An isolated node is a node that does not have any connection with any other node.
- Freeze Network: The drawing algorithm for the network is dynamic and reacts on the re-positioning network nodes. The dynamic behaviour can be turned off so that the nodes of the network can be positioned manually without interfering of the drawing algorithm.
- Time selection: The time slot can be limited to examine the evolvement and state of the network at different points in time.

Furthermore, the feature provides the possibility to save the network visualization as a screenshot in png. Thus, the network can be used in reports or presentations.



RF72 - GUI of the feature

## Implementation details

The feature is implemented as java application together with the repository feature RF65 which generates the network representation of the blogosphere.

Nodes are represented as objects that have an ID and a maximum of five details. Links are an aggregation of one or more Linkevents. A linkevent has one sender, one or more recipients, a timestamp, and a maximum of five details. The distinction of Linkevents with timestamps allows representing and analysing the network evolution over time.

The application uses the following Java classes to represent the network structure:

- *tub.dataElements.Network*
- *tub.dataElements.Node*
- *tub.dataElements.Linkevent*
- *tub.dataElements.Link*

The following classes represent the network graph that is used for the visualisation:

- *tub.aladin.graphNode*
- *tub.aladin.graph*

The GUI of the network visualization is performed by the classes *tub.aladin.showGraphPanel* and *tub.aladin.showNetworkGraphFrame*.

The class *tub.aladin.settings* contains the color configuration of the network visualization.

<b>Implemented By</b>	TUB
-----------------------	-----

Table A.41: Implementation Description: RF72

<b>Feature ID</b>	RF73 (Repository Feature 73)
<b>Name</b>	The archive recommends blogs to users based on the ratings and preferences
<b>Effort Spent</b>	1 Week
<b>Modules Affected / Created</b>	BibRank, WebSearch

### Description of the new feature

This feature introduces a new ranking method to rank the records by their weighted averages.

A portalbox which shows the Top Rated Records has been added into main page.

**Top Rated Records**

- 3.3459 **Question 2 Exam 2**
- 3.3321 Exam 1 Question 1
- 3.3309 **BULLYING-RESTORATIVE JUSTICE**
- 3.3294 Catering for a difference websites
- 3.3289 Basic Assumption (Dreikurs)
- 3.3248 **EDUCATION FOR THE 21ST CENTURY-WORKING WITH GROUPS**
- 3.3233 A poke with a sharp stick | by david carter-tod
- 3.3126 A Difference
- 3.3081 Exactly what is Emotional Intelligence?
- 3.3074 Glasser and Classroom management.

RF73 - Portalbox displaying top rated records

A portalbox which shows last added records has been added into main page.

**Last Added Records**

- 2013-05-24 17:35 **The Hyper-Social Organization**
- 2013-05-24 17:34 **How to Position Enterprise 2.0**
- 2013-05-24 17:34 **Potential Benefits of Enterprise 2.0**
- 2013-05-24 17:34 **What will power next-generation businesses?**
- 2013-05-24 17:34 **Three Secret Weapons Of Innovation**
- 2013-05-24 17:34 **Telstra's 3Rs of Social Media Engagement**
- 2013-05-24 17:34 **Tactics for improving customer engagement**
- 2013-05-24 17:34 **Gary Hayes's social media counter**
- 2013-05-24 17:34 **Google Wave Versus the Rest, Feature by Feature**
- 2013-05-24 17:34 **A CMO's Guide To The Social Media Landscape**

RF73 - Portalbox displaying recently added records

## Implementation details

- Portal boxes have become associated with ranking methods.
  - `bibrank_portalbox` table, which keeps which ranking method is related with which portal box, has been inserted into database.
  - `update_bibrank_portalbox` and `drop_bibrank_portalbox` functions have been added into `bibrank_record_sorter.py` .
    - \* `update_bibrank_portalbox` function updates the portal boxes when bibrank is run.
    - \* `drop_bibrank_portalbox` function removes the entries related to given bibrank method when either the ranking method is deleted or the variable keeping the number of records shown in portalbox is set to `0` .
  - For portalboxes `bibrank_portalbox.html` template is added.
- Ranking with “Weighted Average” has been introduced.
  - To rank the records by their weighted average, `bibrank_weighted_average_indexer.py` module has been created. This module contains the function `weighted_average_to_index` which calculates the weighted average with “Bayesian estimate” which is the following formula:
 
$$\frac{N}{N + m} * A + \frac{m}{N + m} * G$$
 where
    - \* **N**: the number of reviews of corresponding record
    - \* **m**: minimum number of reviews required to calculate the rank of the record
    - \* **A**: Average score of corresponding record
    - \* **G**: Average score of all of the records
  - `bibrank_weighted_average_template.cfg` containing the parameters for the ranking method has been created. These parameters are:
    - \* `show_relevance` : `1` to show the score on search page, `0` otherwise.
    - \* `minimum_review_number` : minimum number of reviews required to be ranked
    - \* `display_on_portalbox_count` : the number of the records will be displayed on the portalbox. If it is `0` , portalbox disappears and entries related to that ranking method is removed from database.
- Archived content indexer has been added as a ranking method to create “Recently Added Records” portalbox.

<ul style="list-style-type: none"> <li>– <b>bibrank_archived_content_indexer.py</b> has been introduced to rank the records in a time interval.</li> <li>– <b>template_recently_archived_content.cfg</b> containing the parameters for the ranking method. These parameters are: <ul style="list-style-type: none"> <li>* <b>latest_records_number</b> : the number of the lastly added records, if <b>0</b> , ranks all of them.</li> <li>* <b>date_type</b> : <b>creation</b> for ranking by creation date, <b>modification</b> for ranking by modification date.</li> <li>* <b>start_date</b> : the beginning of the time interval.</li> <li>* <b>end_date</b> : the end of the time interval.</li> <li>* <b>interval</b> : the sql like time interval. (i.e. <b>3 HOUR</b> , <b>1 DAY</b> )</li> <li>* <b>display_on_portalbox_count</b> : the number of the records will be displayed on the portalbox. If it is <b>0</b> , portalbox disappears and entries related to that ranking method is removed from database.</li> </ul> </li> <li>• Some modifications have been occurred in <b>bibrank.py</b> and <b>bibrank_tag_based_indexer.py</b> to accommodate new ranking method.</li> </ul>		
<table border="1" style="width: 100%;"> <tr> <td style="width: 30%;"><b>Implemented By</b></td> <td>SRDC</td> </tr> </table>	<b>Implemented By</b>	SRDC
<b>Implemented By</b>	SRDC	

Table A.42: Implementation Description: RF73

<b>Feature ID</b>	RF78 (Repository Feature 78)
<b>Name</b>	The archive displays content after filtering it with user preferences
<b>Effort Spent</b>	2 Days
<b>Modules Affected / Created</b>	WebSearch

#### Description of the new feature

The repository offers to users the possibility of personalizing their searches with some options such as: the collections where users wish to search and the number of records to be displayed per page.

#### Implementation details

The search user preferences has been enriched with a multiselect where the user can select the collections that will be taken into account in his searches.

Also the file *websearch\_blueprint.py* has been amended in order to propagate the selected preferences.

## Edit

---

Results per page

Hotkeys

Collections

---

<b>Implemented By</b>	CERN
-----------------------	------

Table A.43: Implementation Description: RF78

<b>Feature ID</b>	RF87 (Repository Feature 87)
<b>Name</b>	The archive transforms the SIPS received from the spider to AIPS
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	BibArchive
<b>Description of the new feature</b>	
<p>New module to archive the AIP (Archival Information Package) of a record. A daemon checks periodically for new or modified records, creates an AIP and stores it in a dedicated database. In the web interface, the detailed view of a record offers the user a link to download the AIP.</p>	

<b>Implementation details</b>	
<p>A new module called BibArchive has been implemented: The database used is MongoDB. It supports versioning, but the old versions of an AIP can only be retrieved from the command-line interface. The AIP is wrapped using BagIt and then zipped. The zipped file is stored in the db directly. It includes:</p> <ul style="list-style-type: none"> <li>• The MARC and METS xml files of the record</li> <li>• All the attached files (mostly images)</li> <li>• The available metadata and md5 checksums of each file</li> </ul> <p>A new tasklet has been created to update the database. The administrator can decide how often this tasklet runs (depending on the upload/modification rates, a value between 30 mins and 1 day is recommended). When it runs, it first retrieves then list of records that have been created or modified since the last time it run. Then, for each them, an AIP is created using BagIt and the content mentioned above and inserts it in the archival database.</p>	
<b>Implemented By</b>	CERN

Table A.44: Implementation Description: RF87

<b>Feature ID</b>	RF88 (Repository Feature 88)
<b>Name</b>	The archive stores the content of the AIPS in two different databases for preservation purposes
<b>Effort Spent</b>	1 Week
<b>Modules Affected / Created</b>	BibUpload, BibIngest
<b>Description of the new feature</b>	
<p>The AIPs are stored in two different databases. In one of them the SIP is stored as received from the spider using BibIngest, and a second copy is used as a working copy and stored in the Invenio metadata database.</p>	
<b>Implementation details</b>	
<p>Details on how this is done can be found in RF9 and RF87.</p>	
<b>Implemented By</b>	Raquel Jiménez, Jaime García, Nikolaos Kasioumis (CERN)

Table A.45: Implementation Description: RF88

<b>Feature ID</b>	RF89 (Repository Feature 89)
<b>Name</b>	The archive carries out the normalization and/or migration of the media attachments
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	BibSched
<b>Description of the new feature</b>	
<p>New tasklet that creates new version of the file of a record, migrating them from one format to another. It accepts plugins to transform the files</p>	
<b>Implementation details</b>	
<p>A new plugin system has been implemented allowing the administrators to implement their own format conversion code and place the files under <code>modules/bibdocfile/lib/format_migration_plugins/</code> Then, the variable <code>CFG_FORMAT_MIGRATION_PLUGINS_MAPPING</code> has to be updated to reflect which plugin is desired to be used for each format.</p> <p>A new tasklet has been implemented to migrate the format of attached files. It retrieves the list of records uploaded or modified since the last time the tasklet run and applies the appropriated conversion plugin according to the variable <code>CFG_FORMAT_MIGRATION_PLUGINS_MAPPING</code>. Finally, it uploads the new file into the record as a new format of the same file, without replacing it.</p>	
<b>Implemented By</b>	CERN

Table A.46: Implementation Description: RF89



## A.3 Features not retained

In order to concentrate on blog-specific must-have repository features, some decisions have been taken regarding nice-to-have feature requirements identified in D4.1[4] user survey. The list of non-retained features is presented below.

The following features were not retained because the complexity of the topic is falling out of scope of blog archiving platform. The solutions developed outside of platform could be integrated with the blog archive in later stages.

- RF42 - The archive extracts bibliographic metadata from content embedded in blogs
- RF68 - The archive provides information diffusion analysis mechanisms
- RF75 - The archive can do sentiment analysis on the content

The following features were not retained because sufficient alternative approaches exist:

- RF50 - The archive offers the option to disseminate newly archived content in external social platforms (alternative: instead of pushing to social platforms, the apps can pull from the blog archive platform via customised RSS feeds)
- RF49 - The archive distinguishes institutional/corporate blogs from personal blogs (alternative: user tags blogs as corporate upon submission, see RF31 description in Section 3.2 of D4.5[8])

The following features were not retained because of low demand and high complexity of the topic:

- RF76 - The archive detects content originality and ranks it accordingly

## Appendix B

# Final Spider Implementation Descriptions

One of the objectives of this deliverable is to present the implementation activities of the whole suite of spider features defined in D4.2[9]. In order to include the description of the features along with its implementation details, the same template that was created in D4.5[8] will be used.

<b>Feature ID</b>	SFID (Spider Feature ID)
<b>Name</b>	One sentence clear enough to make someone who has already read the specification remember the description
<b>Effort Spent</b>	Actual implementation time (Possible values: Days/Weeks/Months)
<b>Modules Affected / Created</b>	Name of the modules either modified or introduced
<p><b>Description of the new feature</b></p> <p>High level description of the feature</p> <p><b>Implementation details</b></p> <p>Technical details of the implementation activities are described here. All the files and modules that are exposed to modifications (i.e adding/ altering classes/methods, introducing new fields into configuration files, new user interfaces, etc) and how they are modified are explained in detail. Screenshots of new functionalities are provided here.</p>	
<b>Implemented By</b>	Person or partner who implemented the feature

Table B.1: Implementation Description: SFID

Section B.1 presents the implementation descriptions of the whole list of spider features that were defined in D4.2[9], and Section B.2 lists those spider features that was decided not to implement, as well as the reasons why this decision was taken.

## B.1 List of implementation descriptions

In this Section the list of implementation descriptions of the final spider features is presented.

<b>Feature ID</b>	SF1 (Spider Feature 1)
<b>Name</b>	Capture timestamps for creation and harvesting
<b>Effort Spent</b>	3 Days
<b>Modules Affected / Created</b>	The capture module, the parser and the extractor
<b>Description of the new feature</b>	
<p>Automatically identify timestamp within the HTML using background knowledge for each blog, post and comment.</p> <p>Initial capture of the RSS identifies the timestamp of the published post as well of the other data elements available in the RSS. This information is used when harvesting the URI of the HTML version of the post.</p> <p>Matching algorithms and machine learning algorithms can also be used to find RSS and time stamp from an inserted blog URI.</p> <p>XML output:</p> <ul style="list-style-type: none"> <li>• Time stamp of the published blog post</li> <li>• The time harvesting the post is also stored and presented in the output – XML and integrated into the repository.</li> </ul>	
<b>Implementation details</b>	
<p>The algorithm uses the date/time tag in the RSS feed to be used as reference in the brute force process for matching against the candidates in the harvested HTML.</p> <p>The candidates are selected using Microsoft date/time parsing algorithm to identify potential candidates.</p>	

This is being combined with using matching technique (Levenstein) to compare text in RSS and HTML.

This knowledge is reused for following analysis for future learning understanding other sites date time format, as well as increase the learning speed.

The output format of time stamp is represented in the XML following the MARC-standard.

<b>Implemented By</b>	CyberWatcher
-----------------------	--------------

Table B.2: Implementation Description: SF1

<b>Feature ID</b>	SF2 (Spider Feature 2)
<b>Name</b>	Retrieve and Parse semi-structured information from the blog
<b>Effort Spent</b>	3 Months
<b>Modules Affected / Created</b>	Host Analyzer

#### Description of the new feature

Detect and extract blog entities such as post, comments and sub entities title, content, tags, authors etc.

Detecting semi-structured information from the blog post – is essential to the spider and follows all part of the harvesting process.

Initial capture of the RSS identifies the timestamp of the published post as well of the other data elements available in the RSS. This information is used when harvesting the URI of the HTML version of the post.

#### Implementation details

Ontology representation of how a blog structure is used as a basis for mapping RSS feed item towards the HTML harvesting and parsing.

Through comparing the text from the RSS items with the HTML version, using modified version of the Levenstein algorithm, we get xpaths1 which both represent the absolute path and a weighting of the different attributes within the DOM structure. This weight will cope with the possibility of different absolute positions as well as class / id values.

By creating several different rules and comparing those, it is then possible to identify common patterns in a single blog.

Using matching algorithms such as Levenstein and machine learning algorithms the RSS data elements are controlled and new identified – such as:

- Blog post headline
- Blog post full text
- Blog author
- Time stamp of the published blog post
- Blog domain
- Blog comments
- Blog pictures
- Blog Tags
- Blog post tags

<b>Implemented By</b>	CyberWatcher
-----------------------	--------------

Table B.3: Implementation Description: SF2

<b>Feature ID</b>	SF3 (Spider Feature 3)
<b>Name</b>	Capture tags of blogs and blog posts
<b>Effort Spent</b>	(Included in SF2)
<b>Modules Affected / Created</b>	Host Analyzer, Spider Core
<b>Description of the new feature</b>	
<p>Extracting tags related to the blog sources and blog posts and storing them. These tags represent or categorizes the content and source.</p> <p>This feature captures tags at three levels:</p> <ul style="list-style-type: none"> <li>• Blog domain</li> <li>• Blog source (watchpoint) such as sections with posting or comments</li> <li>• Blog post or comment (categorized separately)</li> </ul> <p>The different tags for blog domain and watchpoints includes:</p> <ul style="list-style-type: none"> <li>• Host URI (URL identifier for the posts) group blog posts towards same domain</li> </ul>	

- URI (e.g. feed URI) — how to re-find the blog and watchpoint
- Content type HTML text or images
- Character encoding: UTF8 — using standardized characters
- Name Actual name of the source or domain
- Description — actual text describing the source or domain
- Language code (en, US, No, Fi)
- Generator: Technical Platform (as below — which Wordpress version)
- Thumbnail of the source

For the post or comment:

- URI of the post
- URI of the watchpoint
- URI of the domain
- Author
- Technical platform
- Title of the post
- Text of the post
- Links
- Embedded pictures including format e.g. jpg

### Implementation details

Extracting tags from Posts and blogs

- **Post:** When the post div is located, most of the blog system uses standard structures for displaying tags. By mapping these structures the system will extract the tags related and is then stored within the post entity object.  
E.g. `‘‘Posted: Monday, March 11th, 2013 at 4:00 pm. Tags: fashion, Fashion Week’’`
- **Post:** Another type of tags related to the blog post may be microtags — available within the text.
- **Blog domain:** The initial learning of the site will search for given structures on the root page.  
E.g. `HTML/head/meta[@name=keywords]/@content`  
E.g. `<meta content="Bakovervendt, fremovervendt, barnesete, bilstol, beltestol, pute, lovlig, isofix, test, best,`

<pre>sikring, barn, bagnett, boxette, bilsyk, bilsyke" name="keywords"/&gt;</pre> <p>The data is then stored within the entity format describing either the blog or post. This is then converted into XML and MARC for exporting.</p>	
<b>Implemented By</b>	CyberWatcher

Table B.4: Implementation Description: SF3

<b>Feature ID</b>	SF4 (Spider Feature 4)
<b>Name</b>	Support different kinds of blog platforms and blogging software
<b>Effort Spent</b>	Included in SF2
<b>Modules Affected / Created</b>	Host Analyzer / Spider Core

### Description of the new feature

The spider is able to extract content from different blog content software and platforms like Wordpress and blogger.com.

Within each platform, eg Wordpress there are multiple versions of the platform which can have implications of the flexibility of how to set up and design a new blog, as well as features and elements on the blogs.

The spider is generic and not customized for specific platforms. As described in SF2 this is based upon analyzing RSS and matching this with harvested HTML version.

However, new platforms might have different ontology and require to add new rules to the spider. The final spider have been tested towards thousands of blogs representing an unknown number over platforms and versions.

### Implementation details

The spider identifies the ontology of the blog through harvesting the DOM structure from the web version. The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML.

Through comparing the text from the RSS items with the HTML version, using modified version of the Levenstein algorithm, we get xpaths which both represent the absolute path and a weighting of the different attributes within

the DOM structure. This weight will cope with the possibility of different absolute positions as well as class/id values.

By creating several different rules and comparing those, it is then possible to identify common patterns in a single blog – independent of the spider platform.

<b>Implemented By</b>	CyberWatcher
-----------------------	--------------

Table B.5: Implementation Description: SF4

<b>Feature ID</b>	SF5 (Spider Feature 5)
<b>Name</b>	Harvesting of text and related HTML
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	Host Analyzer / Spider Core

#### Description of the new feature

Ability to capture more text than displayed in RSS, such as full text for the blog post and all HTML of the blog post and other HTML element of each blog page.

This feature utilize RSS with the Levenstein matching algorithm and ID3 learning algorithm to compare text and tags in the RSS with the matching text paragraphs in the blog post harvested by the spider from the blog page.

The RSS text is normally shorter than full text – so this feature shall detect the pattern of the text in RSS with text in post and find end of text. Often there is a tag displaying end of text, but sometimes this requires more analyzing than one would expect from seeing the actual blog post.

Also when matching the text, there are sometimes differences due to inconsistent use of RSS-tags, changes in version of text in post and RSS.

This feature captures at three levels:

- Blog domain
- Blog source (watchpoint) – such as sections with posting or comments
- Blog post or comment (categorized separately)

#### Implementation details

Identification of full text in the HTML version includes identifying:



Tags in HTML of where text is ending
Change in DOM structure after text matching towards the RSS
<b>Implemented By</b>   CyberWatcher

Table B.6: Implementation Description: SF5

<b>Feature ID</b>	SF6 (Spider Feature 6)
<b>Name</b>	Harvesting of the comments of blog posts
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	Host Analyzer / Spider Core

### Description of the new feature

This features covers the ability to capture comments in a blog source and identify which blog post they relate to.

Comments are mostly found in separated RSS.

Initial process starts with finding the RSS which contains comments, and identifying which of the two alternative ways of structuring the comments are used:

1. Centralized RSS for comments - contains multiple comments from different posts
2. RSS per post – for each post there is an RSS with all comments for this specific post.

### Implementation details

For centralized RSS the spider uses the same technique as described in SF5 in order to identify the fulltext version of the comment.

In order to map which blog from which post the comment belong, the spider search the RSS for URI relating to blog post. This is the identifier as a “post-source” of the comment. The comments belonging to a single post needs to be tagged with this post-source in order to be connected in the repository.

In addition the spider needs to find an identifier of the first to the last comment. This is handled either through time sorting or number tags in the RSS or the HTML version.

RSS per post can be found when harvesting from the HTML. This type of multiple RSS per blog source requires much more resources to analyze and monitor and from a scaling point of view this has not been supported.

<b>Implemented By</b>	CyberWatcher
-----------------------	--------------

Table B.7: Implementation Description: SF6

<b>Feature ID</b>	SF7 (Spider Feature 7)
<b>Name</b>	Information about the harvesting source
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	Host Analyzer / Spider Core
<b>Description of the new feature</b>	
<p>This feature finds blog source tags and descriptive information about the source relating to all posts.</p> <p>Such tags includes:</p> <ul style="list-style-type: none"> <li>• Host URI (URL identifier for the posts)</li> <li>• Content type — HTML text or images</li> <li>• Character encoding: UTF8 — using standardized characters</li> <li>• Name — Actual name of the source or domain</li> <li>• Description — actual text describing the source or domain</li> <li>• Language code (en, US, No, Fi)</li> <li>• Generator: Technical Platform (as below — which Wordpress version)</li> <li>• Thumbnail of the source</li> </ul>	
<b>Implementation details</b>	
<p>This tags is found through the DOM structure of the blog.</p>	
<b>Implemented By</b>	CyberWatcher

Table B.8: Implementation Description: SF7

<b>Feature ID</b>	SF8 (Spider Feature 8)
<b>Name</b>	Spam detection and filtering
<b>Effort Spent</b>	2 weeks
<b>Modules Affected / Created</b>	Host Analyzer / Web Service/Site / Spider Core
<b>Description of the new feature</b>	
<p>Spam filtering when inserting list of blogs reduces initial spam problems substantially. Major spam issue is dead blogs, blogs which was supposed to be blogs but does not fit or being changed into a non-blog. And also lists may include non-blogs as well just because the list is not correct. In fact it may be a larger challenge to avoid these type of spam, than a typical spam that may be without content at all – or just high frequent volumes of same URL.</p> <p>Instead of using black listing — the spider require all new blog sources to fit into the rules identified with previous blogs in the machine learning algorithm. This way the spider can be very strict or fairly flexible. E.g. new blogs must fulfill all spider rules — or x%.</p> <p>And by visiting the URI we find that the rule setting has disabled this source — actually being a twitter feed site. However, there is a blog section related to the site — but with a different URI: <a href="http://www.kasvi.org/index.php?blog">http://www.kasvi.org/index.php?blog</a></p>	
<b>Implementation details</b>	
<p>Source Analysis: The RSS is initially controlled to contain a minimum of Blog elements, such as title, content and date. If such element is not recognized in RSS — the spider will not harvest the HTML version. When mapping the RSS onto the HTML and no unique identifier has been found the blog is discarded as part of the spam process.</p> <p>User: The user can disable specific blogs, inserting its own black list or import publicly available blacklists – interface for such is not available in the spider and will be done through repository.</p> <p>Ping Monitor: User predefines blog filters, e.g. keywords found in URL and title. Only specific blog URL and titles matching these keywords will pass onto the Analyzer module. This prevents the spider from taking in traditionally large volume spam.</p>	
<b>Implemented By</b>	CyberWatcher

Table B.9: Implementation Description: SF8

<b>Feature ID</b>	SF10 (Spider Feature 10)
<b>Name</b>	Configuration of crawled blogs
<b>Effort Spent</b>	8 weeks
<b>Modules Affected / Created</b>	Host Analyzer / Spider Core
<b>Description of the new feature</b>	
<p>This features includes how to manage the spider as of what to crawl, how to crawl and how to process and export.</p> <p>There are three levels of configuring this:</p> <ol style="list-style-type: none"> <li>1. The end user level</li> <li>2. The admin level</li> <li>3. The central admin level</li> </ol> <p>The user level can be limited to inserting blog URLs.</p> <p>In the admin level the configuration may include polling frequency, limitation of what to spider such as snapshots and thumbnails.</p> <p>In the central admin level — which is only controlled by the developers, new methods can be implemented for improving the spider and its ability to crawl new types of blogs.</p> <p>All levels of configuration is available from a web interface — through the spider web portal. In addition configurations can be done in the web service.</p>	
<b>Implementation details</b>	
<p>All configuration capabilities is described in the web service documentation, Annex 3 in this document and the D 4.6 Final Weblog component, Chapter 3, about the spider web portal.</p>	
<b>Implemented By</b>	CyberWatcher

Table B.10: Implementation Description: SF10

<b>Feature ID</b>	SF11 (Spider Feature 11)
<b>Name</b>	Publishing license type retrieval
<b>Effort Spent</b>	1 week
<b>Modules Affected / Created</b>	Host Harvesting and Analyzer Module

### Description of the new feature

Capturing license agreements regulating usage and access to a blog source.

Harvesting blogs according to policies and issues described in WP3 cannot be implemented into an automated spider.

Initially it is expected that the user and manager of the spider has secured allowance of capturing the blog before inserting the relevant URIs into the spider interface.

The spider has implemented two features to assist in this process of identifying license type and disclaimers and copyright issues:

1. The spider will capture and include information from key identifying terms related to license:
  - Copyright
  - Term of use
  - Terms of use
2. The spider will identify and capture meta-tag for spiders *robots.txt*. This is a standard meta-tag used to inform spiders not to index the site.

### Implementation details

The spider will capture and include information from key identifying terms related to license:

- Copyright
- Term of use
- Terms of use

The spider will identify such links in the blog source or domain — and follow these links to the page where such content can be harvested. This text will then be available as “all text” for the repository. Related to this is also

implementation of handling robots.txt. [https://developers.google.com/search-appliance/documentation/68/admin\\_crawl/Preparing#donotcrawl](https://developers.google.com/search-appliance/documentation/68/admin_crawl/Preparing#donotcrawl)

The spider identifies robots.txt and communicates this to the repository. The repository will be where this is then handled according to user instructions — as the user can be the publisher and then wanting to harvest their own blogs despite a robots.txt.

<b>Implemented By</b>	CyberWatcher
-----------------------	--------------

Table B.11: Implementation Description: SF11

<b>Feature ID</b>	SF12 (Spider Feature 12)
<b>Name</b>	Detection and harvesting of embedded content
<b>Effort Spent</b>	
<b>Modules Affected / Created</b>	Host Analyzer / Spider Core

### Description of the new feature

This feature detects embedded elements in the blog — which can include:

- Pictures
- Documents
- Videos

The spider may harvest most embedded elements — but in the final spider its limited what to harvest due to size restrictions.

Pictures and elements with smaller size elements are downloaded and exported to the repository as part of the XML.

### Implementation details

Embedded pictures, icons and other small sizes elements are harvested as part of the HTML and included in the spider content to be exported as part of XML.

The spider detects size and formats. PNG, JPG and other light graphical oriented formats is automatically included.

For other formats — the spider control the size and which element to save according to this code:

For all tags marked as *<object>* which are embedded objects into the html are automatically analyzed and stored as a reference.

The data is then stored within the entity format describing either the blog or post. This is then converted into XML and MARC for exporting.

<b>Implemented By</b>	CyberWatcher
-----------------------	--------------

Table B.12: Implementation Description: SF12

<b>Feature ID</b>	SF13 (Spider Feature 13)
<b>Name</b>	Scalability and real time harvesting
<b>Effort Spent</b>	2 Months
<b>Modules Affected / Created</b>	Application / Web service / Website
<b>Description of the new feature</b>	
<p>The ability to harvest and monitor updates on larger amount of blogs, with the trade off towards depth and resources required to harvesting frequency. Scaling can be done by either installing multiple applications across multiple computers and adjust the amount of threads that should be running at any time.</p> <p>An initial Analyzer learns and identifies blogs based upon previous indexed blogs.</p> <p>After an inserted blog is analyzed and harvested – revisiting and monitoring blogs are substantially quicker. Revisiting blogs using already established rules makes it more than ten folds as quick.</p>	
<b>Implementation details</b>	
<p>Scaling capability is implemented both in optimizing each module of the software and through the architectural structure. Optimizing the software is done through testing the software and logging the time and CPU required for each step of the spider process.</p> <p>The architectural structure scales by supporting to add more back-end spider applications and using the same web interface. When inserting new blog sites, the load of processing shall be evenly distributed across all the back ends.</p>	
<b>Implemented By</b>	CyberWatcher

Table B.13: Implementation Description: SF13

<b>Feature ID</b>	SF14 (Spider Feature 14)
<b>Name</b>	Capture necessary metadata from crawled content
<b>Effort Spent</b>	3 Days
<b>Modules Affected / Created</b>	Host Analyzer / Spider Core
<b>Description of the new feature</b>	
<p>Capture metadata like Micro Formats and other meta tags describing the source and each post.</p> <p>Extracting meta tags includes feature SF3 — extracting tags describing blogs and blog posts.</p> <p>In addition the spider capture Micro formats which can represent multiple sets of additional data, as described in D2.4 WeblogSpider Prototype, Chapter 3.4.1.</p> <p>The final spider capture this and deliver this to the repository without any categorization or further analyzing. This way the repository may utilize this all type of captured Micro Formats without limitations of new formats and change of formats.</p>	
<b>Implementation details</b>	
<p>Extracting and implementing of meta tags and micro formats — is seen</p>	
<b>Implemented By</b>	CyberWatcher

Table B.14: Implementation Description: SF14



<b>Feature ID</b>	SF15 (Spider Feature 15)
<b>Name</b>	UTF8 as the default character encoding
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	Host Analyzer / Spider Core
<b>Description of the new feature</b>	
<p>Character encoding represents a challenge in all harvesting from the web. Encoding according to UTF8 represents a standard way of reading and representing all characters.</p> <p>This function encodes the string data to UTF-8, and returns the encoded version. UTF-8 is a standard mechanism used by Unicode for encoding wide character values into a byte stream. UTF-8 is transparent to plain ASCII characters, is self-synchronized (meaning it is possible for a program to figure out where in the byte stream characters start) and can be used with normal string comparison functions for sorting and such.</p> <p>The spider extract the text from the web – including the encoding used.</p>	
<b>Implementation details</b>	
<p>The UTF8 encoding is implemented with following programming method</p> <p>For all new host analyzed the system will send the byte stream through an opens source library UDE (<a href="http://code.google.com/p/ude/">http://code.google.com/p/ude/</a>), which is ported from Mozilla Universal Charset Detector. Depending on the result from this analyzer and what the web response from the server is compared to what the character mapping in the HTML head is. When the correct type is selected, it will be converted using .Net is internal character encoding converter to UTF8.</p>	
<b>Implemented By</b>	CyberWatcher

Table B.15: Implementation Description: SF15

<b>Feature ID</b>	SF16 (Spider Feature 16)
<b>Name</b>	Harvesting the author
<b>Effort Spent</b>	Included in SF2
<b>Modules Affected / Created</b>	Host Analyzer / Spider Core

Description of the new feature	
<p>Capturing the blog author of each post or the source either as name or email or both.</p> <p>The name of the author of a blog is normally found through the RSS. This is a standard tag in the RSS. This information will then be matched towards the HTML version.</p> <p>If not found through RSS, the spider will harvest tags in HTML-version, in DOM or simple identify email links. The latter is a bit more uncertain as the spider will not know if it's the email of the author or someone else.</p>	
Implementation details	
<p>Extracting author tags from Posts and blogs</p> <ul style="list-style-type: none"> <li>• Post: When the post div is located, most of the blog system uses standard structures for displaying tags.</li> <li>• Within these structures author and email is normally a separate tag. The system will extract these tags and store it according to the structure of rest of the found tags — as described in SF2 and SF3.</li> </ul>	
<b>Implemented By</b>	CyberWatcher

Table B.16: Implementation Description: SF16

<b>Feature ID</b>	SF17 (Spider Feature 17)
<b>Name</b>	Harvesting the disclaimer
<b>Effort Spent</b>	2 Weeks
<b>Modules Affected / Created</b>	Host Analyzer Module
Description of the new feature	
<p>Capture disclaimers describing liability issues of each blog. It is stored and is accessible through the file description on the blog entity.</p>	

<b>Implementation details</b>	
<p>When a new blog site is added, it will be analyzed to extract data such as the disclaimer. By searching for predefined patterns in URLs and HTML structure the disclaimer will be extracted.</p> <p>After identifying the disclaimer, the spider will follow the link to a disclaimer page. This entire page will be store as a file related to the blog entity. When the repository gets the latest batch from the spider API, they will be requesting all file elements and they will be returned as a byte stream.</p>	
<b>Implemented By</b>	CyberWatcher

Table B.17: Implementation Description: SF17

<b>Feature ID</b>	SF18 (Spider Feature 18)
<b>Name</b>	Harvesting links
<b>Effort Spent</b>	2 weeks
<b>Modules Affected / Created</b>	Host Analyzer Module
<b>Description of the new feature</b>	
<p>Detecting and capture external links per blog posts such as video links, presentations, article references, and link farms.</p> <p>Links in the full text of each blog post is captured and handled both as text and as a separate entity.</p> <p>The spider is classifying the link according to the URL and well known formats such as:</p> <ul style="list-style-type: none"> <li>● PDF</li> <li>● Excel</li> <li>● Docs</li> <li>● HTML</li> <li>● JAVA</li> <li>● MPG</li> </ul> <p>The embedded content is also represented with links and will be found through these links in the repository.</p>	

### Implementation details

When extracting all links within the post it will look at what type of link it is, such as:

- `<img src='''>` When link is found within this type of html tag it is obvious that it is an image link.
- `<embed src=''' type='''>` Embedded data, reading the type (if defined) makes it possible to know what it is
- `<iframe src='''>` Iframe, possible to html page
- `<object data='''>` Could be everything from audi, video to apples and documents
- `<a href='''>` When a normal link is found we can look at some identifiers within the link url, e.g.
  - .doc
  - .pdf
  - etc

But in most cases you must access that source to know what it really is.

<b>Implemented By</b>	CyberWatcher
-----------------------	--------------

Table B.18: Implementation Description: SF18

<b>Feature ID</b>	SF19 (Spider Feature 19)
<b>Name</b>	Snapshot of blog versions
<b>Effort Spent</b>	2 weeks
<b>Modules Affected / Created</b>	Host Analyzer Module

### Description of the new feature

Capture the HTML of each blog and makes a low resolution thumbnail and high resolution snapshot.

This is however not done per post due to size constrains and risk of hammering the blog too extensively. Such can be seen as spamming and the spider then being black listed from the source.

The thumbnail includes the whole length of the text. If the text is short, the thumbnails looks like a screen dump. However, if text is long, this features

includes the entire text and the thumbnail will shrink to fit the height into same format. This could be configured differently – as many thumbnails becomes hard to see if text is long. The alternative is to cut text and only have thumbnail more like a screen dump.

### Implementation details

Snapshot and Thumbnails are implemented using SiteShoter from NirSoft ([http://www.nirsoft.net/utils/web\\_site\\_screenshot.html](http://www.nirsoft.net/utils/web_site_screenshot.html)). Using this application you can configure:

- Resolution
- Image type
- Compression
- Browser width and height
- Etc.

If the blog is configured to take snapshot of the blog, it will create one with normal resolution and one that is reduced to 100 x Y pixels, where Y is calculated to get the correct scaling.

<b>Implemented By</b>	CyberWatcher
-----------------------	--------------

Table B.19: Implementation Description: SF19

<b>Feature ID</b>	SF20 (Spider Feature 20)
<b>Name</b>	Purposive list of crawled blog sources
<b>Effort Spent</b>	6 weeks
<b>Modules Affected / Created</b>	Host Analyzer Module

### Description of the new feature

This features includes the implementation of a search engine and log into the spider. The capability can be seen in the spider web portal:

Search field is possible within sources, posts and entities. The log list is seen both as state: number of indexed source/posts/entity and actual listing below with paging.

Holding such log is a major part of the spider but key to be able to manage and sort captured data as well as avoid already crawled data — see SF 21.

Lucene .net is implemented as search engine (<http://en.wikipedia.org/wiki/Lucene>).

### Implementation details

A custom wrapper been created around Lucene to support our need for search/update and delete object that are strongly typed. In this solution we use this wrapper to create the following indices:

- WatchPoint: Search for the state of all entry points for crawling blogs sites
- Host: Search for hosts that has been registered in the system
- UriState: For all added URLs they are put here for processing. The system uses this index to keep track of all the urls the user has added.
- Entity: Search for blogs, posts and comments.

The query syntax is described here:

<http://lucene.apache.org/core/2.9.4/queryparsersyntax.html>

<b>Implemented By</b>	CyberWatcher
-----------------------	--------------

Table B.20: Implementation Description: SF20

<b>Feature ID</b>	SF21 (Spider Feature 21)
<b>Name</b>	List of already crawled posts – separating old from new posts
<b>Effort Spent</b>	Included in SF20
<b>Modules Affected / Created</b>	Host Analyzer Module

### Description of the new feature

The spider holds lists of already crawled blog posts and comments to separate new from old post. This is key both to avoid spamming the source with multiple harvesting of same posts as well as avoid duplicates in the harvesting and storing.

Through using RSS as initial phase of harvesting the spider gets IDs and URI of only the latest posts. Any RSS updates gives new posts only, with no need to harvest other posts.

Second method is matching URI towards spider log. As long as the spider keeps the complete log — the ID represented with URI will detect duplicates and stop harvesting and indexing old posts.

### Implementation details

Through using RSS as initial phase of harvesting the spider gets IDs and URI of only the latest posts. Any RSS updates gives new posts only, with no need to harvest other posts.

Second method is matching URI towards spider log. As long as the spider keeps the complete log — the ID represented with URI will detect duplicates and stop harvesting and indexing old posts.

<b>Implemented By</b>	CyberWatcher
-----------------------	--------------

Table B.21: Implementation Description: SF21

<b>Feature ID</b>	SF22 (Spider Feature 22)
<b>Name</b>	METS packages of crawled content
<b>Effort Spent</b>	2 weeks
<b>Modules Affected / Created</b>	Exporting module, webservice and feed

### Description of the new feature

The captured content has been parsed and structured into XML to be available for repositories. In this project the XML should be formatted according to METS, as seen below.

This is securing the repository being able to understand all entities available from the spider.

### Implementation details

The webservice exposes a get-method where the method transforms the internal XML-representation of the blog entity using XSLT transformation

into a METS document according to the MARC XML format. Each document is described with METS namespace in the XML.

The MARC contains three different structures – one for blog source, blog post and one for blog comments.

Within the XML there is a set of file references (FileGrp) containing metadata describing the content as well as where content can be retrieved; either from the spider server or from the original blog source.

Example of FileGrp:

```
<file ID="2.jpeg" MIMETYPE="image/jpeg" SIZE="25975"
CREATED="29.04.2013 12:51:51">
<FLocat
LOCTYPE="URL">http://indianautosblog.com/wp-content/...</FLocat>
<FContent CHECKSUM="d8a729cbbc4c036279d3b1f14ba98ec2">
</FContent>
</file>
```

The repository can use the webservice to fetch all the METS documents.

<b>Implemented By</b>	CyberWatcher
-----------------------	--------------

Table B.22: Implementation Description: SF22

## B.2 Features not retained

During the development process some features have been considered too resource processing to be fully implemented. Such considerations have been discussed with the rest of the technical BlogForever team to see if there are alternatives and to discuss about the prioritization of those features. In addition to this, such discussions have been taken in the entire BlogForever project meetings — before final decision being made about leaving out such features [2].

Finally, the only spider feature that has not been implemented after such process is:

- SF9 - Information about the context of the blog: Personal vs. business/institutional.

SF9 covers classification of the blog source and its content. Separating end user blogs from professional blogs, e.g. corporates, authors and editor managed blogs. It has not been possible to find an exact way of categorizing blogs by content and context. Just implementing content categorization would require a very resource demanding process for the spider. It would have to text analyzing the related blog post as well as the rest of the posts within the same blog domain for all posts



— or at least absolutely all blog domains. This would take resources away from scaling and other extraction tasks of the spider — and require more hardware. Also, the chance of making a successful categorization — fully automated - across different type of industries, languages and format of blogs — is fairly low. And even more importantly the main usage of the blog spider has been defined as users can insert already known blog URLs into the spider. Then they have the capability to categorize better and more efficient than an automated spider can achieve through text-mining.

## Appendix C

# Spider API Documentation

The spider web service is the main API for enabling repository to utilize the weblog spider for data harvesting. It includes ability to administrate the spider and to allow end users to insert blog URLs in the repository.

The web service is accessible for any technical license holder that wants to set up a separate spider locally or to use the web service as SaaS for connecting towards a dedicated repository.

Each web service is managed through username and password. The login to the web service is available at <http://bf.cyberwatcher.com/>. Username and password can be given through michael@cyberwatcher.com.

In addition to this, there will be implemented API keys for authentication and access control purposes. The API keys are generated through the web interface.

The complete web service API description is presented below.

The API consists of 15 methods:

- Version
- AddWatchPoint
- DeleteWatchPoint
- UpdateHost
- DeleteHost
- GetWatchPoint
- GetEntity
- SearchEntities
- SearchWatchPoints
- SearchHosts
- SearchLog
- UpdateWatchPoint
- GetDocumentStorage

- GetDocument
- GetDocumentAsMets

The description of each method is presented below with a summary, the input parameters and the return value:

```
[ServiceContract]
public interface IServiceApi
```

```
<summary>
```

```
Returns the given version for connected to the given API key
```

```
</summary>
```

```
<param name="apiKey">Authentication key</param>
```

```
<returns>Returns the version number of the given backend
```

```
Version information for an assembly consists of the following four values:
```

```
Major Version
```

```
Minor Version
```

```
Build Number
```

```
Revision
```

```
</returns>
```

```
[OperationContract]
```

```
string Version(string apiKey);
```

```
<summary>
```

```
A method for adding blog urls, the return objects describes the status of the given url added, if the url has already been added, that record is returned with.
```

```
</summary>
```

```
<param name="apiKey">Authentication key</param>
```

```
<param name="watchPoints">array of urls pointing to the blogsite</param>
```

```
<param name="tags">It is a feature not currently in use </param>
```

```
<returns>Returns the state of the given url, the record contains state as well as messages and what the forward url is. The different states are:
```

```
* New, the url is new and just added
```

```
* Unknown, the url has been processed, but the system could not identify it as a valid blog
```

```
* Processing, the url is being analyzed
```

```
* Invalid, either no rules were created or the site does not exist/blocked etc
```

```
* Entity, rules have been created and the system
```

```
* Forward, if the url is forwarded it is defined here
```

```
* Queued, the url is queued
```

```
* Finished, successful download of blog entities
```

```
* Failed, by some reason it has failed, it will try at a later time
```

```
</returns>
[OperationContract]
UriState[] AddWatchPoints(string apiKey, string[] watchPoints, string[] tags);
```

```
<summary>
A method for deleting a watchpoints
</summary>
<param name="apiKey">Authentication key</param>
<param name="watchpointId">The ID of the watchpoint to be deleted</param>
<returns>True or false if it where successful</returns>
[OperationContract]
bool DeleteWatchPoint(string apiKey, int watchpointId);
```

```
<summary>
A method for updating the host
</summary>
<param name="apiKey"></param>
<param name="hostDescription">Object contains several different fields
that are possible to change, OverrideUserAgent, State and name</param>
<returns>Returns the updated host description</returns>
[OperationContract]
SystemHostDescription UpdateHost(string apiKey, SystemHostDescription
hostDescription);
```

```
<summary>
Method for deleting the host
</summary>
<param name="apiKey">Authentication key</param>
<param name="hostId">Id of the host</param>
<returns>Returns true if successful</returns>
[OperationContract]
bool DeleteHost(string apiKey, int hostId);
```

```
<summary>
Returns a description of the watchpoint , it contains information such
as taking snapshot of the site, override useragent etc.
</summary>
<param name="apiKey">Authentication key</param>
<param name="watchpointId"></param>
<returns>Returns a description / configuration of the given watchpoint</returns>
[OperationContract]
WatchPointDescription GetWatchPoint(string apiKey, int watchpointId);
```

```
<summary>
A method for getting a specific entity, an entity is either:
```

- Blog
- Post
- Comment

</summary>

<param name="apiKey">Authentication key</param>

<param name="entityId">Id of the entity</param>

<returns>Returns a blog entity</returns>

[OperationContract]

BlogEntity GetEntity(string apiKey, int entityId);

<summary>

The method gives the possibility to search for entities using lucene search syntax, for more information: [http://lucene.apache.org/core/old\\_versioned\\_docs/versi](http://lucene.apache.org/core/old_versioned_docs/versi)  
To get an overview of what fields are available perform an empty search

</summary>

<param name="apiKey">Authentication key</param>

<param name="request">Search request object</param>

<returns>Returns a blog search response object containing data such as time spent, search result, facets, fields etc</returns>

[OperationContract]

BlogSearchResponse SearchEntities(string apiKey, SearchRequest request);

<summary>

The method gives the possibility to search for watchpoints using lucene search syntax, for more information: [http://lucene.apache.org/core/old\\_versioned.doc](http://lucene.apache.org/core/old_versioned.doc)

</summary>

<param name="apiKey">Authentication key</param>

<param name="request">Search request object</param>

<returns>Returns a watchpoint search response object containing data such as time spent, search result, facets, fields etc</returns>

[OperationContract]

WatchpointSearchResponse SearchWatchPoints(string apiKey, SearchRequest request);

<summary>

The method gives the possibility to search for hosts using lucene search syntax, for more information: [http://lucene.apache.org/core/old\\_versioned\\_docs/versi](http://lucene.apache.org/core/old_versioned_docs/versi)

</summary>

<param name="apiKey">Authentication key</param>

<param name="request">Search request object</param>

<returns>Returns a host search response object containing data such as time spent, search result, facets, fields etc</returns>

[OperationContract]

HostSearchResponse SearchHosts(string apiKey, SearchRequest request);

<summary>

The method gives the possibility to search for hosts using lucene search syntax, for more information: [http://lucene.apache.org/core/old\\_versioned\\_docs/versi](http://lucene.apache.org/core/old_versioned_docs/versi)

</summary>

<param name="apiKey">Authentication key</param>

<param name="request">Search request object</param>

<returns>Returns a log search response object containing data such as time spent, search result, facets, fields etc</returns>

[OperationContract]

LogSearchResponse SearchLogs(string apiKey, SearchRequest request);

<summary>

A method for updating the watchpoint description, if successful the object is returned, not all fields are allowed to be modified

</summary>

<param name="apiKey">Authentication key</param>

<param name="watchpoint">The the changed object</param>

<returns>Returns the modified description object</returns>

[OperationContract]

WatchPointDescription UpdateWatchPoint(string apiKey, WatchPointDescription watchpoint);

<summary>

Returns an objects containing an overview of all files connected to this specific document id

</summary>

<param name="apiKey">Authentication key</param>

<param name="documentId">Document Id / Entity Id</param>

<returns>Returns a storage info object containing file association descriptions</returns>

[OperationContract]

StorageInfo GetDocumentStorage(string apiKey, int documentId);

<summary>

Returns the given binary file associated

</summary>

<param name="apiKey">Authentication key</param>

<param name="documentId">Document id / entity id, owner of the files</param>

<param name="filename">The file that is to be returned</param>

<returns>Returns the given binary file associated</returns>

[OperationContract]

byte[] GetDocument(string apiKey, int documentId, string filename);

<summary>

Converts from the internal format (xml) to a METS document structure

</summary>

```
<param name="apiKey">Authentication key</param>
<param name="documentId">Document id / entity id</param>
<returns></returns>
[OperationContract]
MetsDocment GetDocumentAsMets(string apiKey, int documentId);
```

```
<summary>
A method of check what state a given url that has been added is
</summary>
```

```
<param name="apiKey">Authentication key</param>
<param name="uri">The already added url</param>
<returns>Returns the state of the url</returns>
[OperationContract]
UriState GetUriState(string apiKey, string uri);
```

```
<summary>
```

```
The method gives the possibility to search for uristates using lucene
search syntax, for more information: http://lucene.apache.org/core/old\_versioned\_docs
</summary>
```

```
<param name="apiKey">Authentication key</param>
<param name="request"></param>
<returns>Returns a search result set of type UriState</returns>
[OperationContract]
UriStateSearchResponse SearchUriState(string apiKey, SearchRequest request);
```

```
<summary>
```

```
Restarts the watchpoint if for some reason it has failed.
```

```
</summary>
```

```
<param name="apiKey">Authentication key</param>
<param name="id"></param>
<param name="reCreateRules">If the system should be forced to also delete
the rules and recreate them</param>
[OperationContract]
void ResetWatchPointById(string apiKey, int id, bool reCreateRules);
```

```
<summary>
```

```
Resets a given entity, it should be downloaded and analyzed once more
```

```
</summary>
```

```
<param name="apiKey">Authentication key</param>
<param name="id"></param>
[OperationContract]
void ResetEntityById(string apiKey, int id);
```

## Appendix D

# Deployment Instructions

## D.1 Repository Deployment Instructions

This Section presents the deployment guide to install the Repository component.

### D.1.1 Software Requirements

The software requirements needed to run the BlogForever Repository are either Linux packages or Python packages. The Linux packages will be installed automatically in the installation process described in the next subsection. The commands that do this part are:

```
sudo -u www-data make install-jquery-plugins
sudo -u www-data make install-jquery-tokeninput
sudo -u www-data make install-jstimeline-plugin
sudo -u www-data make install-mathjax-plugin
sudo -u www-data make install-mediaelement
```

Regarding the Python packages, the source files include several files listing the packages needed. These files are `requirements.txt`, `requirements-extras.txt`, `requirements-flask.txt`, and `requirements-flask-ext.txt`. In Debian, for example, it is easy to use these files. The following commands will install the packages using the appropriated version of each of them.

```
sudo pip install -r requirements.txt
sudo pip install -r requirements-extras.txt
sudo pip install -r requirements-flask.txt
sudo pip install -r requirements-flask-ext.txt
```

### D.1.2 Instructions

The code of the BlogForever repository is divided in 2 different git code repositories:



- One of them is a new branch on top of Invenio's `next` branch and has the features developed for BlogForever that can be used in Invenio (they are generic to any digital repository). The name of this branch is `invenio-blogforever-next`. We will refer to it as “the Invenio side”.
- The second one is a different repository where you can find the features developed that are not compatible with the general-purpose version of Invenio. This repository can be found in <http://invenio-software.org/repo/blogforever/> and the latest branch is called `next`. We talk about it as “the BlogForever side”

Therefore, you will need 2 different directories with the source files. From now on, we will assume `/src/invenio` and `/src/blogforever`.

To bring the latest version of the BlogForever side under the right directory:

```
cd /src
git clone http://invenio-software.org/repo/blogforever/
cd blogforever
git fetch
git checkout next
```

To bring the latest version of the Invenio side under the right directory:

```
cd /src
git clone http://invenio-software.org/repo/invenio/
cd invenio
git remote add
  jgarcial http://invenio-software.org/repo/personal/invenio-jgarcial
git fetch jgarcial invenio-blogforever-next
git checkout invenio-blogforever-next
```

Now you have all the latest source files. Let's install all of it. First, we install the Invenio side. According to the Invenio install instructions, you do:

*This is needed only the first time*

```
cd /src/invenio
aclocal-1.9
automake-1.9 -a
autoconf
./configure
```

*These 2 lines are only ones needed every time you reinstall the code*

```
sudo make
sudo -u www-data make install
```

*This is needed only the first time*

```
sudo -u www-data make install-jquery-plugins
sudo -u www-data make install-jquery-tokeninput
sudo -u www-data make install-jstimeline-plugin
sudo -u www-data make install-mathjax-plugin
sudo -u www-data make install-mediaelement
pip install -r requirements.txt
pip install -r requirements-extras.txt
pip install -r requirements-flask.txt
pip install -r requirements-flask-ext.txt
```

Let's install the BlogForever side sources:

```
cd /src/blogforever
make
sudo -u www-data make install
```

Now, all the source files are in the right directories (by default `/opt/invenio`). We need to load the config variables and restart apache:

*These 2 commands will also be needed every time you reinstall the sources*

```
sudo -u www-data /opt/invenio/bin/inveniocfg --update-all
sudo /etc/init.d/apache2 restart
```

Let's create the database and fill it with the default values:

```
sudo -u www-data /opt/invenio/bin/inveniomanage database create
sudo -u www-data /opt/invenio/bin/inveniocfg --create-demo-site
```

And there you go. Your site will be up and running... hopefully. I guess you also want some records. For this, you will need to get your BibSched ready: This will tell the scheduler to start running the tasks in the queue. You can also stop it.

```
sudo -u www-data /opt/invenio/bin/bibsched start
```

Config variables to be added in your `invenio-local.conf` file:

```
CFG_BIBUPLOAD_FFT_ALLOWED_LOCAL_PATHS = /tmp,/home,/opt
CFG_BIBSCHED_MAX_NUMBER_CONCURRENT_TASKS = 2
CFG_FLASK_SERVE_STATIC_FILES = 1
CFG_BLOGFOREVER_SITE = 1
CFG_FLASK_CACHE_TYPE = redis
CFG_DEVEL_TOOLS = werkzeug-debugger
CFG_FLASK_DISABLED_BLUEPRINTS = webdeposit_blueprint
CFG_WEBSTYLE_TEMPLATE_SKIN = blogforever.css
```

After adding this configuration to your `invenio-local.conf` file you should run the following command so they are taken into account:

```
sudo -u www-data /opt/invenio/bin/inveniocfg --update-all
```

Let's add the fetcher task. It will download data from the spider (BF4 instance in this case). It will upload records into the database:

*Note that this all is just one command*

```
sudo -u www-data /opt/invenio/bin/bibtasklet
-T bst_fetch_records_from_spider
-a url='http://bf4.itc.auth.gr/Spider20130620/SpiderService.svc?wsdl'
-a api_key='DZFbrYM5pm0s6xKafZSz9+1LeJVoiyWhgF3VQU0+4dg='
-a constant_set=20 -u admin
```

This task keeps the id of the last record fetched from the spider. If you want to start fetching all the records from the beginning again, you will need to remove the file where the last id was stored (`/opt/invenio/var/tmp/last_id`) Probably you will also want to remove all the current records. To do that you just need to run the following command:

```
sudo -u www-data /opt/invenio/bin/inveniocfg --remove-demo-records
```

In order to be able to find the records when searching or browsing, you'll need to add other tasks to the BibSched queue:

*These commands will load the basic tasks that will run every 5 minutes*

```
sudo -u www-data /opt/invenio/bin/bibindex -f50000 -s5m -u admin
sudo -u www-data /opt/invenio/bin/bibreformat -oHB -s5m -u admin
sudo -u www-data /opt/invenio/bin/webcoll -v0 -s5m -u admin
sudo -u www-data /opt/invenio/bin/bibrank -f50000 -s5m -u admin
sudo -u www-data /opt/invenio/bin/bibsort -s5m -u admin
```

*These commands will load housekeeping tasks that will run only once a week*

```
sudo -u www-data /opt/invenio/bin/bibrank -f50000 -R -wwrd -s14d
-L Sunday -u admin
sudo -u www-data /opt/invenio/bin/bibsort -R -s7d
-L 'Sunday 01:00-05:00' -u admin
sudo -u www-data /opt/invenio/bin/inveniogc -a -s7d
-L 'Sunday 01:00-05:00' -u admin
```

## D.2 Spider Deployment Instructions

This Section presents the deployment guide to install the Spider component.

### D.2.1 Software Requirements

- **Windows Server** (Preferably version 2008 or newer).
- **.NET Framework** versions 3.5, 4 and 4.5.
- **Microsoft SQL Server 2008**, preferably 2012, (also works with SQL Server express but there is a limit of 4GB to the size of the database).
- Microsoft IIS web server.

### D.2.2 Instructions

#### 1. Microsoft IIS web server

- (a) Install IIS from the add/remove software menu of Windows server.
- (b) Open the **IIS Control panel**, and click on the **Windows Platform Installer Icon**. From there, you are able to install all **.NET** versions as well as the **MVC4** library.
- (c) Enable **Windows Communication Foundation (WCF) Library**.
- (d) Enable **HTTP activation**.
- (e) All **.NET** versions should have the same settings.

#### 2. Microsoft SQL server

- (a) Install the Microsoft SQL server following standard installation procedure (CDROM installation setup).
- (b) Use the **BlogForever Spider database deployment script** `deploy.sql` which creates the initial database structure.

#### 3. BlogForever Spider Web component

- (a) Extract the BlogForever spider zip file in the IIS web folder (`c:/inetpub/wwwroot/Spider` by default).
- (b) Open the IIS control panel, right click on the folder and select **Convert to Application**. (All the files beneath this folder are parts of a web application, if not it is not accessible).
- (c) Setup database connection. Edit the **web.config** file inside the Spider web folder, containing the database connection string which has to be modified according to server settings.

#### 4. BlogForever Web Crawler component

- (a) Extra the BlogForever Crawler zip file in a selected folder.

- (b) Note that The default file storage location of the spider executable is in a subfolder below its location.
- (c) Edit **CW.CrawlerSystem.ConsoleApp.exe.config** file to setup database connection string
- (d) Furthermore, inside the AppSettings there are important application variables.
  - i. **EntityStorage** is the folder location of the entities the crawler extracts,
  - ii. **SourcesStorage** is the location of the sources description,
  - iii. **WebRequestCache** is used to cache every HTTP request result, expiration time is 10min (this is necessary for HTTP traffic optimisation)
  - iv. **UserAgent** is a copy of the Opera UserAgent,
  - v. **mexHttpBinding** is the description of all the network services that this program provides,
  - vi. **baseAddress** it the address of mexHttp
  - vii. **SvcConfigEditor** is used to define new endpoint connections towards a service. You can use the same application to configure your client towards a service or the other way round. Example of service description is in this address: <http://bf4.itc.auth.gr/Spider/SpiderService.svc>
  - viii. `maxStringContentLength` and `maxString` should not be modified.
- (e) Execute the **CW.CrawlerSystem.ConsoleApp.exe** to run the crawler.

