



1.2 SEVENTH FRAMEWORK PROGRAMME
FP7-ICT-2009-6
BlogForever
Grant agreement n^o.: 269963

D4.5: Initial Weblog Digital Repository Prototype

Revision:	First Version
Dissemination Level:	Public
Author(s):	Şenan Postacı, Alper Çınar, Gokce Banu Laleci, Jaime García Llopis, Raquel Jiménez Encinar, Vangelis Banos, Apostolos Papadopoulos, Ilias Trochidis, Konstantinos Davarinos, Lamprini Kolovou, Christos Kontas, Panagiotis Chatzikamaris
Due date of deliverable:	January 31, 2013
Actual submission date:	October 22, 2013
Start date of the project:	March 01, 2011
Duration:	30 months
Lead beneficiary name:	SRDC Yazılım Arastırma ve Gelistirme ve Danismanlık Ticaret Limited Sirketi (SRDC)

Abstract:

This report presents the implementation activities carried out for the BlogForever digital repository component. In this respect, it provides detailed implementation descriptions as well as the testing activities of the implemented features according to their feature specifications. Furthermore, this report outlines the adopted collaboration workflow together with the technologies utilized.

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 Arastirma ve Gelistirme ve 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
0.1	First draft	SRDC	27/11/2012
0.2	Implementation descriptions of SRDC added	SRDC	04/12/2012
0.3	Suggested improvements	AUTH	10/01/2013
0.5	Second draft	SRDC	21/01/2013
0.6	Suggested improvements	UW, CERN	24/01/2013
0.7	Implementation descriptions of AUTH added	AUTH	24/01/2013
0.8	Implementation descriptions of CERN added	CERN	24/01/2013
0.9	Minor details edited	SRDC	28/01/2013
1.0	Final version	SRDC	29/01/2013

Table of Contents

ExecutiveSummary	1
1 Introduction	2
1.1 Background	3
2 The Weblog Digital Repository Implementation	4
2.1 BlogForever and Invenio	4
2.2 Implementation	6
2.3 Software Testing	9
2.3.1 Unit Tests	10
2.3.2 Regression Tests	10
2.3.3 Web Tests	10
2.4 User Testing	11
3 Implementation Descriptions	14
3.1 Features already in Invenio	15
3.2 List of implementation descriptions	17
4 Conclusions and Future Work	76
References	78

List of Figures

1.1	BlogForever Platform overview	2
2.1	Invenio Overall Architecture	5
2.2	Digital Repository Overall Architecture	6
2.3	Branch Workflow in Invenio	8
2.4	Git Workflow in BlogForever	9
2.5	Feature Testing Illustration [6]	12
2.6	Iterative Development Cycles [6]	13
3.1	RF1 - Sample dashboard box	17
3.2	RF1 - Configuration box	17
3.3	RF2 - A sample item on “Your History” page	18
3.4	RF2 - The activity detail as tooltip	19
3.5	RF2 - History filter panel	19
3.6	RF3 - Sharing activity panel	20
3.7	RF17 - Disclaimer	24
3.8	RF22 - “Recommended for you” box	25
3.9	RF24 - Links to other sources menu	27
3.10	RF25 - Tags	28
3.11	RF31 - Submit a blog	30
3.12	RF31 - Modify a blog	30
3.13	RF31 - Delete a blog	31
3.14	RF31 - Delete a post	31
3.15	RF32 - Personal data removal interface	33
3.16	RF32 - Registration failure message	33

3.17	RF32 - “Welcome back” message	33
3.18	RF47 - How to cite box for blogs	37
3.19	RF47 - How to cite box for blog posts	38
3.20	RF47 - How to cite box for comments	38
3.21	RF48 - A translated record	39
3.22	RF48 - Translate link in a message	39
3.23	RF48 - A translated content can be reverted to its original	39
3.24	RF57-58-61 - New ranking method on adding ranking method admin interface	42
3.25	RF57-58-61 - Ranking methods on BibRank admin interface	42
3.26	RF57-58-61 - Ranking methods on advanced search panel	42
3.27	RF57-58-61 - Ranking by record view number	43
3.28	RF57-58-61 - Ranking by average score	43
3.29	RF59 - METS option in main search page	45
3.30	RF59 - METS option in detailed record page	45
3.31	RF59 - Record exported as METS	45
3.32	RF62 - PDF and JPEG as export options	46
3.33	RF62 - Jpeg of a record	46
3.34	RF64 - External login providers	49
3.35	RF64 - Adding an application to FourSquare	50
3.36	RF64 - Client ID and Secret obtained from FourSquare	51
3.37	RF70 - Adding new premium package panel	56
3.38	RF70 - Monitoring premium packages in admin panel	57
3.39	RF70 - A restricted collection that requires premium package to display . .	57
3.40	RF70 - Warning message indicates that the collection is restricted	57
3.41	RF70 - Available premium packages to display current collection	58
3.42	RF70 - A form to purchase a premium package with credit card	58
3.43	RF70 - The error message states that the credit card information is wrong .	58
3.44	RF70 - Confirmation page that also displays the premium packages the user have	59
3.45	RF70 - PayPal express checkout screen	59

3.46	RF70 - PayPal transaction confirmation page	59
3.47	RF70 - Confirmation page that also displays the premium packages the user have	60
3.48	RF70 - “Your Account” page displaying current premium groups the user joined	60
3.49	RF70 - The admin panel displaying payment history	60
3.50	RF70 - The admin panel displaying current users joined premium groups .	61
3.51	RF70 - Ogone purchase with credit card from	63
3.52	RF70 - PayPal purchase with credit card from	64
3.53	RF71 - The icon to activate highlighting	67
3.54	RF71 - Color palette	67
3.55	RF71 - Highlighting a part of text	68
3.56	RF71 - The icon to edit highlighted text	68
3.57	RF71 - The edit menu of an highlighted text	68
3.58	RF71 - Adding annotation for a text	68
3.59	RF71 - An annotated text	69
3.60	RF71 - Displaying/editing an annotation	69
3.61	RF71 - The icon to retrieve color palette	69
3.62	RF73 - Ranking with weighted averages of the records	72
3.63	RF73 - Portalbox displaying top rated records	72
3.64	RF73 - Portalbox displaying recently added records	72

List of Tables

3.1	Implementation Description Template	14
3.3	Implementation Description: RF1	18
3.4	Implementation Description: RF2	19
3.5	Implementation Description: RF3	20
3.6	Implementation Description: RF4	21
3.7	Implementation Description: RF5	22
3.8	Implementation Description: RF6	22
3.9	Implementation Description: RF9	23
3.10	Implementation Description: RF12	24
3.11	Implementation Description: RF17	24
3.12	Implementation Description: RF22	26
3.13	Implementation Description: RF23	26
3.14	Implementation Description: RF24	27
3.15	Implementation Description: RF25	28
3.16	Implementation Description: RF26	29
3.17	Implementation Description: RF28	29
3.18	Implementation Description: RF31	32
3.19	Implementation Description: RF32	34
3.20	Implementation Description: RF35	35
3.21	Implementation Description: RF40	35
3.22	Implementation Description: RF41	36
3.23	Implementation Description: RF45	37
3.24	Implementation Description: RF47	38

3.25	Implementation Description: RF48	40
3.26	Implementation Description: RF53	40
3.27	Implementation Description: RF54	41
3.28	Implementation Description: RF57-58-61	44
3.29	Implementation Description: RF59	45
3.30	Implementation Description: RF62	48
3.31	Implementation Description: RF64	55
3.32	Implementation Description: RF67	56
3.33	Implementation Description: RF70	66
3.34	Implementation Description: RF71	71
3.35	Implementation Description: RF73	74
3.36	Implementation Description: RF87	75
3.37	Implementation Description: RF88	75

Executive Summary

The BlogForever platform consists of two main software components: the spider component and the digital repository component. This document intends to describe the implementation process of the digital repository component prototype performed within the scope of *Task 4.5 Implementation of the weblog digital repository web application component* (T4.5) of *Work Package 4 Software Infrastructure* (WP4).

The BlogForever digital repository component is based on Invenio¹, which is a digital archiving system developed and maintained by CERN. The main goal of this deliverable is to explain how Invenio is enhanced and customized for blog preservation by implementing a set of features to fulfill BlogForever project requirements. For digital repository, a total of 89 features, each having a certain task to accomplish, were identified from a list of user requirements, and designed in *T4.4 Design of the digital repository component*. In general, these features enhances Invenio from the following aspects:

- Blog rendering
- Blog metadata
- Blog tagging
- Spider communication
- Ingestion
- Spam filtering
- Social features
- New export options
- Billing system

Implemented features are tested and validated through a combination of test cases designed and implemented in *Work Package 5 Case Studies* (WP5) with the contribution of project members as well as third parties, including bloggers other stakeholders. Another goal of this deliverable is to briefly explain testing activities carried out in WP5 for repository features.

¹<https://invenio-software.org/>

Chapter 1

Introduction

The primary objective of the BlogForever project is to find solutions for aggregation, preservation, management and dissemination of blogs. To achieve these goals, the BlogForever project aims to develop a software platform that enables real-time harvesting of blog entities and preservation of these blogs to facilitate extensive search and exploration functionalities of the archived blogs.

The software architecture behind the intended repository system consists of two main components - the weblog spider (in the following document it will be referred to just as “spider”) and the digital repository. The spider is responsible for crawling all the necessary blog data and capturing/extracting characteristics designated for preservation, while the digital 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. The overall architecture is illustrated in Figure 1.1.



Figure 1.1: BlogForever Platform overview

BlogForever Task 4.5 Implementation of the digital repository component has been designated to develop the digital repository component based on Invenio but specially tailored for weblog needs. Details regarding the outcome of this task are presented in this deliverable.

Finally, in the following text, the words *weblog* and *blog* will be interchangeably used to describe the same concept.

1.1 Background

The BlogForever Description of Work (DoW)[1] 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”.

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. Taking into CERN’s participation in BlogForever project, Invenio software suite is selected for the basis of the digital repository component.

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*[3]. Based on these requirements, 89 repository features, to be built on top of Invenio in order to develop a complete digital repository for blogs, were identified and architecture of the digital repository were described in detail in *D4.4 Digital Repository Component Design*[4].

Implementation of a repository feature defines a required modification of Invenio modules or development of new module(s) to be introduced into Invenio. Therefore implementation phase has followed an incremental, iterative and agile approach, yielding the final digital repository component. The digital repository component will be integrated with the spider component developed in *Task 4.3 Implementation of the weblog spider component* (T4.3) into the final BlogForever platform during T4.6.

Last but not least, implementation activities performed in WP4 have been evaluated in WP5. To be more specific, implemented features have been tested and validated during *Task 5.2 Implementation of the case studies* based on the 6 case studies designed in *D5.1 Design and Specification of Case Studies*[6]. Since implementation phase adopts an agile approach, testing phase also adopts principles of agile testing which require testing to be an integral part of the software development.

The organization of this deliverable is as follows: Implementation and testing activities of the digital repository component carried out within the scope of T4.5 are described in Chapter 2. Chapter 3 introduces the *Implementation Description* concept, and lists the status of the repository features together with their implementation descriptions. Finally, this deliverable is concluded in Chapter 4.

Chapter 2

The Weblog Digital Repository Implementation

In this section, the importance of Invenio for the BlogForever project is presented. In addition, implementation activities of the digital repository component performed within the scope of WP4 are described. These activities include necessary modifications and additions to Invenio at the code level as well as software and user testing to validate the execution of the implemented features, and details regarding them are presented in the following sections.

2.1 BlogForever and Invenio

Rather than implementing a repository for blog preservation from scratch, which is quite time-consuming, the BlogForever project has enhanced Invenio's features and applied necessary modifications in order for Invenio to answer blog needs and provide a comprehensive repository for blog preservation, management and dissemination. Therefore, Invenio is the core of the digital repository component which is an integral part of the final BlogForever platform.

Invenio is an open-source suite of applications for digital library management, developed and maintained by CERN, and covers all aspects of digital library management from document ingestion through classification, indexing, and curation to dissemination. It is currently in use in CERN and many other scientific institutions for document archiving.

Invenio is built on top of a modular architecture enabling it to be flexible and customizable. Each module in Invenio performs a certain task, works independently and may collaborate with other modules. Tasks of each module are already explained in D4.4, so they are not included in this deliverable. Additionally, having support for open standards, such as MARC21¹ and Open Archives Initiative protocol for metadata harvesting (OAI-PMH)², enables interoperability with external digital libraries and nominates Invenio as a complete solution for digital archiving. Following figure illustrates the relationship among modules and how they are organized.

¹<http://www.loc.gov/marc/bibliographic/>

²<http://www.openarchives.org/pmh/>

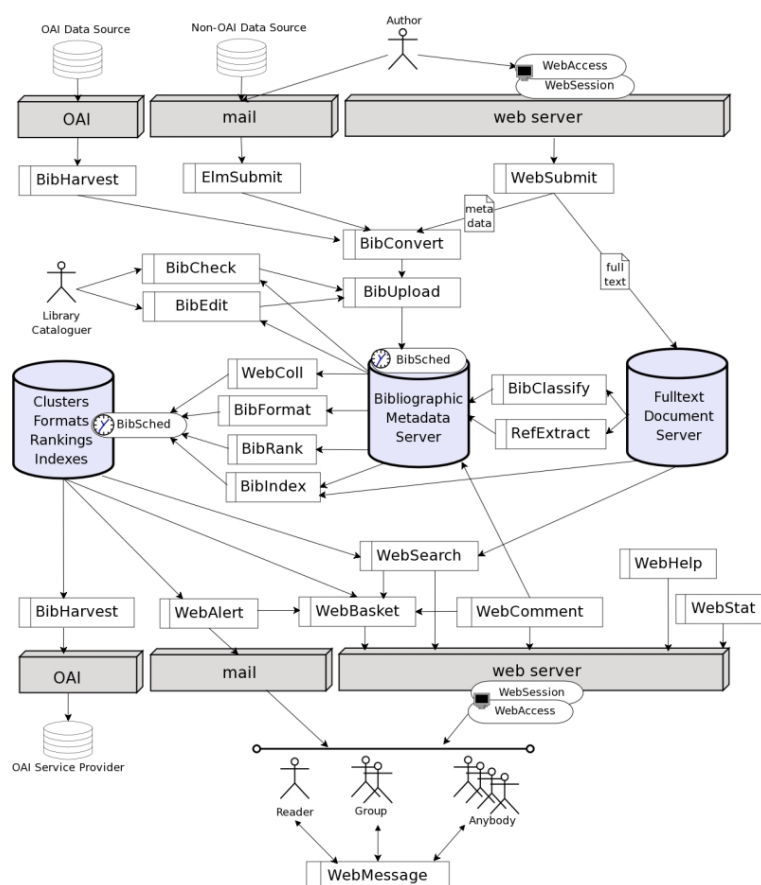


Figure 2.1: Invenio Overall Architecture

From a more technical aspect, Invenio runs on a GNU/Linux³ system, utilizes an Apache/Python⁴ web server and MySQL⁵ database server. It is written in Python⁶ programming language, with some ad hoc modules developed in C⁷ and Common Lisp⁸. Moreover, web technologies such as JavaScript⁹, AJAX¹⁰, JQuery¹¹ and etc. are also utilized to add more functionality.

As the design document of the digital repository component D4.4 clearly states, 89 features were identified to fulfill the requirements for the final digital repository component. In general, these features describe how Invenio is extended and specify functionalities that enable an efficient storage for blog preservation and offer rich user interaction such as advanced weblog related information retrieval, managing a customizable dashboard and socializing by building a user community.

To be more specific, repository features denote the following extensions to Invenio:

³<http://www.gnu.org/gnu/linux-and-gnu.en.html>

⁴<http://code.google.com/p/modwsgi/>

⁵<http://www.mysql.com>

⁶<http://www.python.org>

⁷<http://www.open-std.org/jtc1/sc22/wg14>

⁸<http://www.common-lisp.net/>

⁹<https://developer.mozilla.org/en-US/docs/JavaScript>

¹⁰<https://developer.mozilla.org/en/docs/AJAX>

¹¹<http://jquery.com/>

- **Blog rendering** for representation of four main record types: Blog, Blog Post, Page and Comment
- **Blog metadata** to define blog specific properties and extend MARC schema used in Invenio
- **Spider communication** for communication between the spider and the digital repository
- **BibIngest module** for ingestion of submitted material.
- **WebTag module** for enabling users to tag blogs
- **Spam filtering** for evaluation of the aggregated blog content.
- **Social features** for dissemination of blog content in external social platforms and socialization of users in the platform
- New **export options** such as METS¹², PDF and JPEG
- **Billing system** for exploitation of added value services

In the light of the abovementioned extensions, following figure represents the overall architecture of the BlogForever repository design:

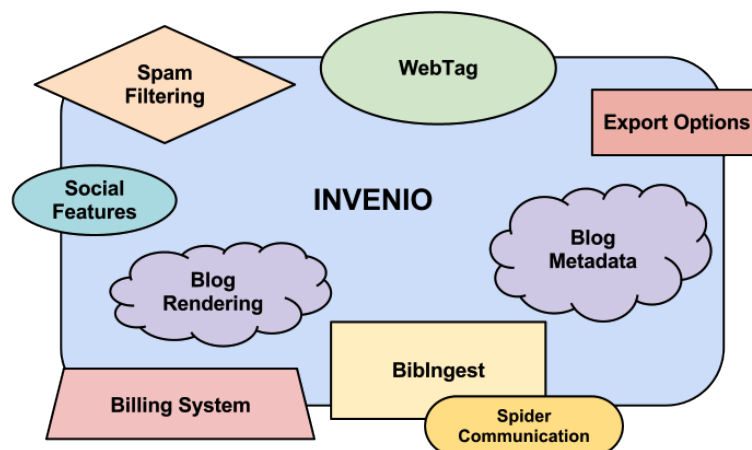


Figure 2.2: Digital Repository Overall Architecture

Implementation of these extensions on Invenio is crucial since without these extensions, this project cannot provide a solution for blog preservation in the long-term, which is the main objective embraced by the BlogForever project. Implementation activities carried out within the scope of T4.5 and the methodology adopted for the development and evaluation of the digital repository component are explained in the following subsections.

2.2 Implementation

BlogForever Deliverable 4.4 Digital repository component design provides also specifications for repository features, describing the functionality of each from an observer point of view. In this way, expected behaviour of a feature is defined and the effort that is needed to implement the feature is estimated before actual coding and test case generation. Therefore, these specifications are considered as reference points during the

¹²<http://www.loc.gov/standards/mets/>

implementation phase. The main aim of the implementation phase conducted during T4.5 is to develop these features according to their specifications to achieve a complete digital repository built on top of Invenio for blog preservation.

The digital repository component development team consists of software developers from code contributor partners in the BlogForever consortium. Since the digital repository component is based on Invenio, a great deal of knowledge about it is essential for development. For this purpose, CERN organized a 3-day hands-on training workshop for developers to get acquainted with Invenio in June 2011. In this workshop, inner details of Invenio regarding its core components and their tasks as well as the collaboration workflow adopted for Invenio development were presented.

After developers from code contributor partners got familiar with Invenio, repository features were distributed among these partners. Therefore, each code contributor partner is responsible for implementation and documentation of the features that were assigned to them.

For code contribution, Git¹³, which is a distributed revision control and source code management (SCM) system, is used since Invenio developers use Git to submit their code to Invenio repositories. Git enables distributed development and provides strong support for non-linear (branching and merging) development, which is very important and essential for Invenio development.

Invenio has an official Git repository located at CERN servers and it contains the official and stable releases. Moreover, the Invenio development organization has a notion of personal local vs. remote repositories. Since Git enables distributed development, each developer has a local copy of the entire development history. After developers change the code in their local copy, they send their local changes to their personal remote repositories, which are located at CERN servers as well, by using Git. More about Invenio repositories can be found at “Invenio git repositories”¹⁴ page.

A CERN account is needed to enable external developers to have access to CERN repositories via SSH. Therefore, CERN accounts were created for each software developer who contributes to the development of the digital repository component.

The collaboration model[2] that is followed in Invenio development adopts a *feature based workflow* that suggests creation of a new branch for each new feature. When a new feature is decided to be integrated, a new development branch, which denotes a slightly different direction in which development is proceeding, is created. After the feature is implemented, the development branch is sent to developer’s personal remote repository and later merged with the master branch of the official repository by the head developer if the feature is complete. Therefore, a development branch will never affect a stable release unless someone explicitly merges those changes into the stable release.

Currently, official Invenio repository has various branches for maintenance and development. The most important three branches can be described as follows[2]:

- **master** branch is where the new features are being developed and where the new feature releases are being made from. The code in master is reviewed and verified, so that it should be possible to make a new release out of this branch almost at any

¹³<http://git-scm.com/>

¹⁴<http://invenio-software.org/repo>

given point in time. If a new feature is well implemented, tested and considered stable, it goes directly into the *master* branch

- **maint** branch is used for maintenance of the latest stable release and contains code having a bugfix nature.
- **next** branch contains code that is cleaned, tested and almost stable, but not fully *master* worthy yet. The *next* branch serves as a kind of stabilisation branch for *master*.

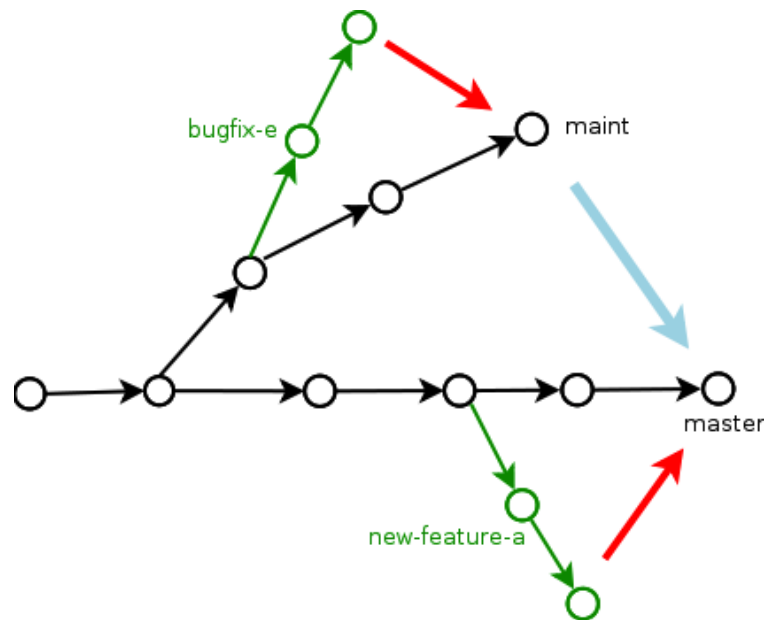


Figure 2.3: Branch Workflow in Invenio

For the development of the digital repository component, a new Git repository, named *blogforever*, was created by CERN and initially contained a copy of Invenio. The whole development history can be tracked at “BlogForever official git repo”¹⁵ page.

Since the digital repository is based on Invenio, same tools and technologies described in Section 2.1 are utilized and development of this component adopts exactly the same workflow described above. In fact, it is already stated in DoW such that “the implementation of the digital repository will follow an organic growth development model. The planned modifications to the vanilla Invenio source code will be implemented through iterations. During these iterations, a new modification or add-on will be implemented, tested and documented each time.”

Therefore, iterations will focus on the following directions:

- Implementation of new modules within Invenio, to match requirements specific to weblog archiving, and not covered by existing software
- Customisation and adaptation of existing Invenio modules to match at best users expectations when navigating/searching the BlogForever web interface

An *iteration* corresponds to implementation of a repository feature and starts with either *cloning* (git clone) the latest release of Invenio or updating the *master* branch to

¹⁵<http://invenio-software.org/repo/blogforever/>

the latest version (`git pull`) into developer's local repository. Then, the developer creates a new branch for the feature, changes the related modules of Invenio or adds new modules according to the feature specification and *commits* (`git commit`) all those changes to his local repository. After that, the developer writes various tests to validate the execution of the feature and commits them, as well (See Section 2.3 for more details). Finally, the developer *pushes* (`git push`) the new branch into his personal remote repository, and this branch is revised and merged by CERN with the master branch of official BlogForever repository. The iteration, ends after the documentation of the new feature is completed. Documentation is performed through implementation descriptions and the details of the repository features with their implementation descriptions can be found in Chapter 3.2.

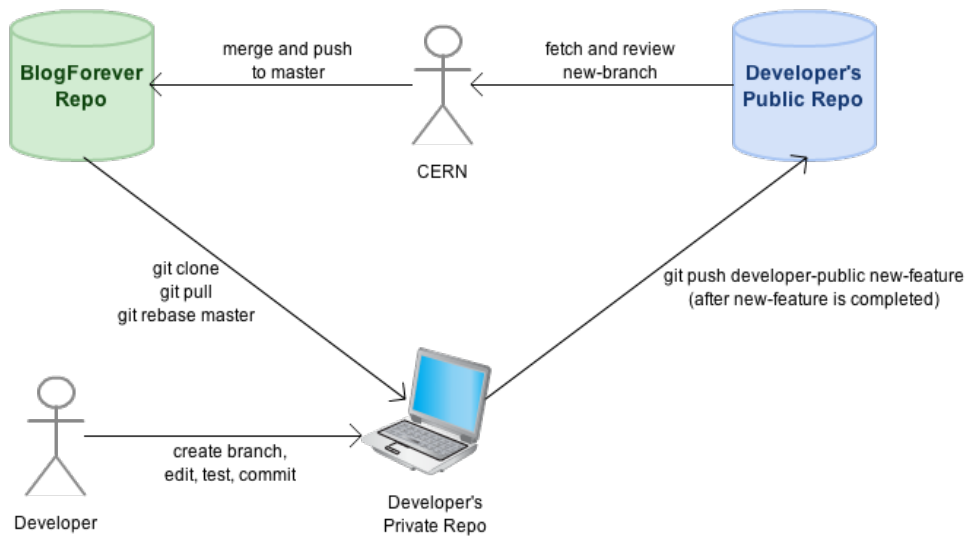


Figure 2.4: Git Workflow in BlogForever

The current version of the BlogForever Weblog Digital Repository source code has been uploaded in the blogforever.eu project portal in the following location: <http://blogforever.eu/wp-content/plugins/wp-filemanager/incl/libfile.php?&path=%2F1.%20Project%20documents%2FSubmitted%20Deliverables%2FD4.5.InitialWeblogDigitalRepositoryPrototype%2F&filename=repository.zip&action=download>

2.3 Software Testing

Software testing is an important complementary process during implementation and performed to reveal potential bugs and failures after a feature is implemented. During software testing, the behaviour of the feature is compared against the expected behaviour defined in its feature specification. The digital repository utilizes various automated testing tools for generation and execution of several test cases in order to ensure that a feature is implemented successfully in the code level. These tools are explained in the following sections.

2.3.1 Unit Tests

Unit testing refers to tests that verify functionalities of a specific section of the source code. For the digital repository, a unit refers to a class together with its methods and unit tests involve test cases for classes that provide important functionalities. Therefore, it is not necessary to write unit tests for each class or its methods. It is also important to note that unit tests are written free of side-effects, i.e., without altering any parts of database.

After implementing a feature, test cases that cover typical as well as corner case inputs are prepared. To explore all possibilities, generation of all combinations of test cases for each method of a class is also preferable.

Since digital repository is mostly written with Python programming language, unit tests are written with Python's testing framework. Test files have the same name of the class to be tested with a suffix `_unit_tests`, e.g., unit test file of `search_engine.py` is `search_engine_unit_tests.py`.

Unit tests are executed by either calling a global executable or launching a specific test file.

2.3.2 Regression Tests

Regression testing focuses on discovering defects emerged after a significant code change. More specifically, it ensures that behaviour of the code is not altered after modifications and the change has not introduced new faults.

Writing regression tests for each feature is an essential step for implementation of the features due to the fact that completed development branches of implemented features are integrated into the BlogForever official repository. Moreover, since the features are being implemented concurrently by several partners, regression tests have a key role in understanding interoperability of the features and finding potential bugs arised during integration.

As in the unit testing case, regression tests are written with Python's testing framework, as well. Test cases include formerly resolved bugs and interactions between different modules. In addition, Python testing framework also provides a module, which is named as *mechanize* and enables simulation of web browsers, and it is used to test overall behaviour of the web pages after integration. Although advanced web technologies such as JavaScript and AJAX can not be tested with this module, it can post form inputs, follow links and surf on the pages.

Regression test files have the same name with the module or file to be tested with a suffix `_regression_tests`. Since regression tests may cause side effects, such as altering database content, tests are executed on a testing environment, e.g on a demo site, etc.

2.3.3 Web Tests

Web testing is required for the features that provide user interfaces based on JavaScript and JQuery where Python's *mechanize* module used in regression testing is ineligible.

Web tests are closer to the end user environment since they involve testing web pages with utilizing a real browser. During the implementation of digital repository, Firefox¹⁶ web browser with Selenium IDE extension¹⁷ is used to automatize web tests. Selenium¹⁸ is a web application testing tool and enables automation of various web browsers. Selenium IDE, on the other hand, is a Selenium project implemented as a Firefox extension and enables recording, editing, and debugging tests. Moreover, it provides a practical way to write web test cases such as surfing on the web pages, sending AJAX requests, testing JQuery components and etc.

Naming of web test files follows the similar convention, i.e. web test files have a suffix `._web_tests` following the related file or module name.

2.4 User Testing

BlogForever project aims to design and implement six case studies within the scope of WP5 in order to evaluate the final BlogForever platform infrastructure. These case studies will be both generic and domain specific, and will ensure that the implementation of the platform is successful. Moreover, the impact of the digital repository will also be evaluated by monitoring system usage and gathering user feedback.

D5.1 lists six case studies with the number of target blogs and their domains as follows:

ID	Nature	Number of blogs	Domain and content
I	Small & simple	28	Higher & Further Education UK
II	Small & simple	70	Higher & Further Education UK
III	Small & complex	356	Multilingual
IV	Small & complex	1,000	Multimedia
V	Large & complex	2,000	Wide ranging
VI	Large & complex	500,000	Wide ranging

Each case study has a target set of features for testing and validation. Therefore, potential problems will be resolved through the different case studies. Certain characteristics, which are based on these features, offered by the final BlogForever platform are validated in WP5 in order to ensure that they are implemented successfully in WP4. Hence, collaboration between WP4 and WP5 is essential and this collaboration is illustrated in Figure 2.5. Case studies focus on the following characteristics of the final BlogForever platform:

- Digital preservation
- Data migration
- Quality assurance checking
- Collection curation
- Additional national laws

¹⁶<http://www.mozilla.org/en-US/firefox/central/>

¹⁷<http://seleniumhq.org/projects/ide/>

¹⁸<http://seleniumhq.org/>

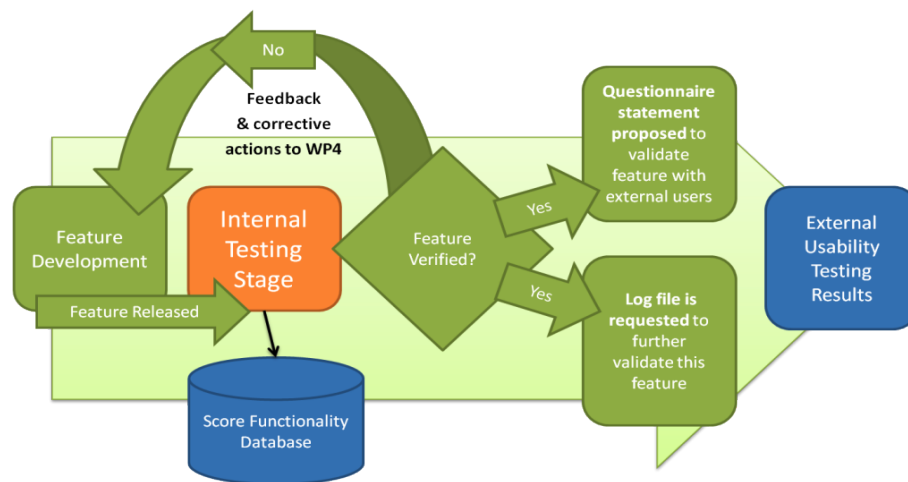


Figure 2.5: Feature Testing Illustration [6]

Implementation of the case studies takes place in two phases. During the first phase, developed components of both spider and digital repository were tested and evaluated so that in case of any problematic issues, they will be fixed at earlier stages. After modifying spider and digital repository in the first phase, the entire system were set up at AUTH Data Center for the second phase. During this phase, case studies will be repeated till the end of the project for longer periods of time, and on larger user groups.

Users play a central role during the implementation of the case studies. Several external users including bloggers and other stakeholders as well as internal project members apart from developers will participate in the testing process, and their feedback will be gathered and analysed. The analysis will help developers to determine if the developed software meets the required results and minimize the system problems. External users will contribute to testing with questionnaires, direct observation and usage of the system and structured interviews, while case studies researchers and software testers will perform the internal assessment of the implementation process. Moreover, external users' interactions with the BlogForever website will be recorded as logs and these logs will be analyzed to identify problems, as well.

D5.1 defines the testing process as *programme of continual improvement* and proposes that software testing should follow an agile, incremental and iterative development cycles. As mentioned earlier, this philosophy is adopted in development process as well. D5.1 also states that "Testing and coding are done incrementally and iteratively, building up each feature to improve the overall outcome". Thus, testing is an integral part of the software development for the BlogForever project and the iterative development cycle can be seen in Figure 2.6 as a whole.

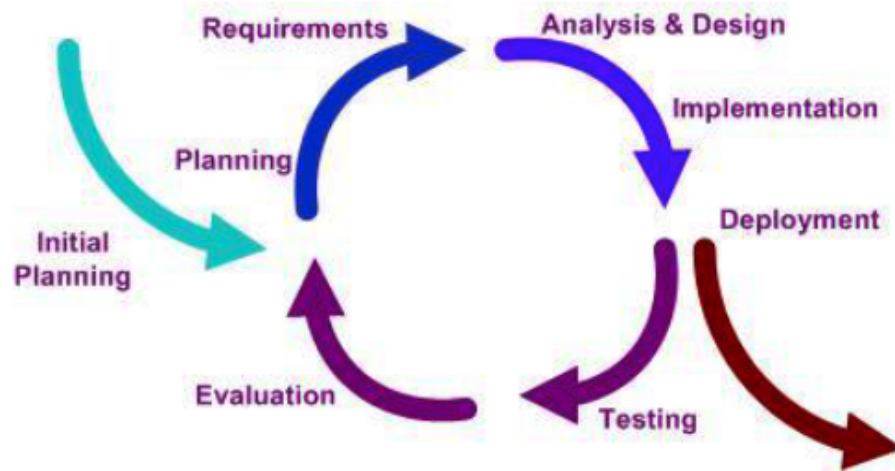


Figure 2.6: Iterative Development Cycles [6]

Chapter 3

Implementation Descriptions

One of the objectives of this deliverable is to describe the implementation activities of each feature. For this purpose, SRDC created a template called *implementation description template*, for documentation of how features are implemented during T4.5. This template is very similar to the *feature specification template* introduced in D4.4, but it is designed to include description of the new feature along with its implementation details.

Feature ID	Short feature identifier: RFXX (Repository Feature XX)
Name	One sentence clear enough to make someone who has already read the specification remember the description
Effort Spent	Actual implementation time (Possible values: Days/Weeks/-Months)
Modules Affected / Created	Name of the Invenio modules either modified or introduced
Description of the new feature	
<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>	
Implementation details	
<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>	
Implemented By	Person or partner who implemented the feature

Table 3.1: Implementation Description Template

It is important to note that, out of 89 repository features, 37 features have been implemented and 31 features are already in Invenio which means that current prototype has 68 features ready and operational, at the moment. After remaining 21 features are implemented, digital repository component will be completed.

Section 3.1 lists the features that are already in Invenio and Section 3.2 presents the implementation descriptions of completed features.

3.1 Features already in Invenio

As mentioned earlier, Invenio is a comprehensive software for digital library management. Therefore, it already supports 31 of the repository features that meet the requirements of the final BlogForever platform. These features are listed as follows:

Feature ID	Feature Name
RF7	Export data using the OAI-PMH protocol
RF8	Export data in Dublin Core schema
RF10	Archive user passwords are stored encrypted in the database
RF11	The web interface is available in many different languages
RF13	UTF-8 is used as the default character encoding in the archive
RF14	Descriptive statistics are offered by record
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 users 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
RF55	The archive provides advanced APIs for developers to interact with the archive's content
RF60	The archive can export all its content, database entries and file system for migration
RF74	The archive enables/disables certain functionalities based on the content rights
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
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

3.2 List of implementation descriptions

Feature ID	RF1 (Repository Feature 1)
Name	Customizable user dashboard
Effort Spent	2 Weeks
Modules Affected / Created	WebSession

Description of the new feature

The current user informative dashboard of Invenio has been upgraded to a customizable version.

For each information element in “Your Account” page (Your Messages, Your Searches, etc...), a box, which is able to be dragged-dropped, expanded-collapsed and created-closed, has been constructed. New templates have been generated for each box.

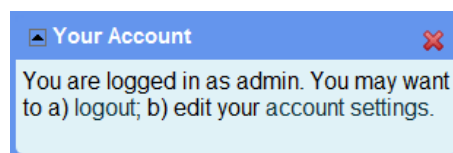


Figure 3.1: RF1 - Sample dashboard box

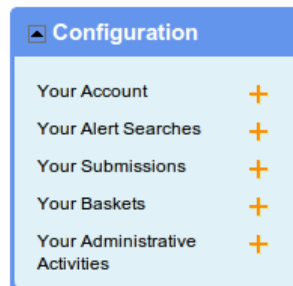


Figure 3.2: RF1 - Configuration box

When the arrow on the top-left corner of the box is clicked, if the box is expanded, it becomes collapsed and vice versa.

The box can be closed by clicking the “X” symbol on the right-top corner of the box. When it is closed, it appears in the “Configuration” box on the left. Also each box can be dragged-dropped by clicking its header and dragging the mouse.

Implementation details

- jQuery is used to provide drag-drop functionality.
- To arrange styles of boxes and “Your Account” page layout, a new CSS file ([inveniodashboard.css](#)) has been created.
- `save` and `insert_new_box` methods were added to `WebInterfaceYourAccountPages` class. `save` method saves the current state of the boxes to the database via an AJAX POST request. It is called after each box action. `in-`

<code>sert_new_box</code> method adds a new box to the page. When clicked to the plus “+” symbol on the “Configuration” box, this method is called via an AJAX POST request.
Implemented By Şenan Postacı, Alper Çınar (SRDC)

Table 3.3: Implementation Description: RF1

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

Description of the new feature

A new “Your History” page is constructed in Invenio. This page includes the user’s history in reverse-chronological order with 16 main categories:

- Downloads
- Comments
- Reviews
- Votes
- Abuse reports
- Subscriptions
- Created groups
- Joined groups
- Viewed records
- Created baskets
- Added items to a basket
- Notes on baskets
- Searches
- Alerts
- Submissions
- Messages

A sample item on “Your History” page can be seen below:

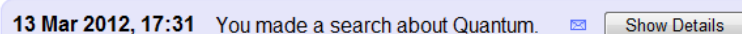


Figure 3.3: RF2 - A sample item on “Your History” page

By clicking “Show Details” button, the activity detail can be seen.

The new box for “Your History” has been created in “Your Account” page. It includes the last 10 activities. By holding cursor over each item, the activity detail is shown as a tool-tip.

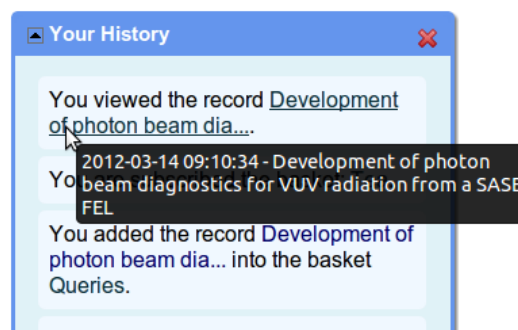


Figure 3.4: RF2 - The activity detail as tooltip

A filter component is developed for filtering activity history.

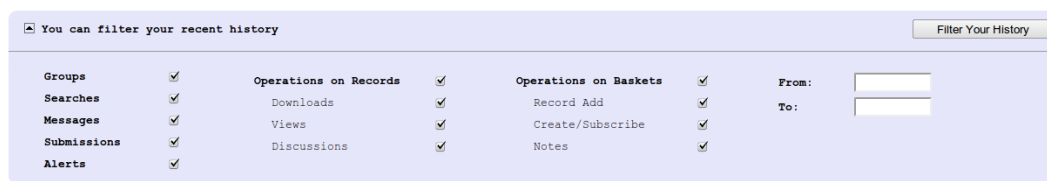


Figure 3.5: RF2 - History filter panel

Based on user's selections, this component can filter activity history for the specified time interval.

Implementation details

- A new class, **WebInterfaceYourHistoryPages**, is created to handle “Your History” page and two new python modules, **webhistory.py** and **webhistory_dblayer.py** which include history-related functions, are implemented.
- Following changes are applied in order to track activities related to baskets:
 - To get the the history of created baskets, **creation_date** and **creation_date** columns are inserted into table **user_bskBASKET**.
 - To get the alerts on specific time, type of the column **date_creation** are changed to datetime from date in the table **user_query_basket**.
- Also **webbasket_dblayer** are changed to accommodate this changes.

Implemented By	Şenan Postacı, Alper Çınar (SRDC)
-----------------------	-----------------------------------

Table 3.4: Implementation Description: RF2

Feature ID	RF3 (Repository Feature 3)
Name	“Share” option in “Your History” box
Effort Spent	2 Weeks
Modules Affected / Created	WebSession

Description of the new feature

Near each activity in the “Your History” and “Your Account” pages a tiny  icon is added.

By clicking this icon, a pop-up message dialog, which is reconstructed on the server for the corresponding activity and sent back, is displayed. The message dialog can be expanded and dragged-dropped.

Autocomplete search can be used to find users or groups.

To send message on a later date, if preferred, “Send Later?” section can be filled by picking a date.



Figure 3.6: RF3 - Sharing activity panel

Implementation details

- To reconstruct activity information, `basket_type`, `convert_activity_header_on_share` and `generate_links` methods in `webhistory.py` are used. The latter method is used for providing links to the receivers if they have access to see corresponding page.
- For both user and group search, jQuery-autocomplete mechanism, that queries keywords on the lists as user types some letters, is utilized.
- For that operation, two new methods are implemented in `webinterface.py` under `WebInterfaceYourHistoryPages` class namely `search_users` and `search_groups`. Each time the user types a letter, these methods are called.
- `send` method in `webmessage_webinterface` is used for sending message to the provided users or groups.
- A new JavaScript file `activity_share.js` and a CSS file `activity_share.css` are created.

Implemented By	Şenan Postacı, Alper Çınar (SRDC)
-----------------------	-----------------------------------

Table 3.5: Implementation Description: RF3

Feature ID	RF4 (Repository Feature 4)										
Name	Bibformat output templates to display blogs and blog posts differently										
Effort Spent	3 Weeks										
Modules Affected / Created	WebAccess, WebSearch, Bibformat										
Description of the new feature											
<p>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:</p>											
<table border="1"> <thead> <tr> <th>Tag content</th> <th>Collection name</th> </tr> </thead> <tbody> <tr> <td>BLOG</td> <td>Blogs</td> </tr> <tr> <td>POST</td> <td>Posts</td> </tr> <tr> <td>COMMENT</td> <td>Comments</td> </tr> <tr> <td>PAGE</td> <td>Pages</td> </tr> </tbody> </table>		Tag content	Collection name	BLOG	Blogs	POST	Posts	COMMENT	Comments	PAGE	Pages
Tag content	Collection name										
BLOG	Blogs										
POST	Posts										
COMMENT	Comments										
PAGE	Pages										
<p>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.</p>											
Implementation details											
<p>The file <code>democfgdata.sql</code> 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.</p>											
Implemented By	Raquel Jiménez, Jaime García (CERN)										

Table 3.6: Implementation Description: RF4

Feature ID	RF5 (Repository Feature 5)
Name	The web interface provides harmonized access and ensures compatibility with major browsers
Effort Spent	1 Week
Modules Affected / Created	BibFormat
Description of the new feature	
<p>The Invenio templating system has been used to harmonize the look and feel of the content.</p>	

Implementation details	
	<ul style="list-style-type: none"> • BibFormat Elements are Python files that given a record return a piece of HTML. It is normally a small piece of information. • BibFormat Templates combine many BibFormat Elements and build a complete rendering of a given record. • Bibformat Output define a series of rules. Given a record, and depending on the content of the MARC tags in the metadata, these rules will determine which BibFormat Template to use. • See RF4 for more details.
Implemented By	Raquel Jiménez, Jaime García (CERN)

Table 3.7: Implementation Description: RF5

Feature ID	RF6 (Repository Feature 6)
Name	Latest posts are displayed sorted by addition date
Effort Spent	1 Day
Modules Affected / Created	MiscUtil
Description of the new feature	
<p>Latest addition lists of posts can be made easily customizable by administrators. This is done by setting up and running one of the plugins defined under the directory “modules/websearch/lib/websearch_instantbrowse_plugins”</p>	
Implementation details	
<p>The plugin <code>websearch_instantbrowse_by_field.py</code> is set up with its corresponding parameters to the blog posts collection in order to display them sorted by publication date. This is configured by default into the initial configuration file called <code>democfgdata.sql</code>.</p>	
Implemented By	Raquel Jiménez Encinar (CERN)

Table 3.8: Implementation Description: RF6

Feature ID	RF9 (Repository Feature 9)
Name	The archive stores and displays accordingly all record metadata received from the spider
Effort Spent	2 Weeks
Modules Affected / Created	BibUpload

Description of the new feature

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.

After the records have been inserted in the repository, the techniques described in RF4 and RF5 are used to display the metadata.

Implementation details

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 **bp_pre_ingestion.py** 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:

- FFT tags are inserted with references to every attached file downloaded from the spider.
- Metadata tags are inserted linking each record to other existing records, like the parent record
- The parent record license and visibility are propagated to the being uploaded record.

Implemented By	Raquel Jiménez, Jaime García (CERN)
-----------------------	-------------------------------------

Table 3.9: Implementation Description: RF9

Feature ID	RF12 (Repository Feature 12)
Name	The archive can import METS
Effort Spent	1 Week
Modules Affected / Created	BibUpload

Description of the new feature

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.

Implementation details

The pre-ingestion plugin is defined in `modules/bibupload/lib/preprocess/bp_pre_ingestion.py` 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.

Implemented By	Jaime García (CERN)
-----------------------	---------------------

Table 3.10: Implementation Description: RF12

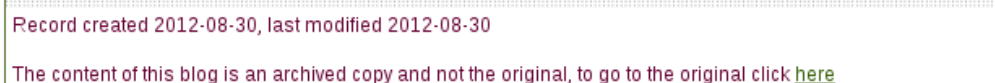
Feature ID	RF17 (Repository Feature 17)
Name	The archive displays a disclaimer about the originality of the content
Effort Spent	2 Days
Modules Affected / Created	BibFormat

Description of the new feature

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.

Implementation details

This is implemented in the `webstyle.template.py` 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.



Record created 2012-08-30, last modified 2012-08-30
The content of this blog is an archived copy and not the original, to go to the original click [here](#)

Figure 3.7: RF17 - Disclaimer

Implemented By	Raquel Jiménez Encinar (CERN)
-----------------------	-------------------------------

Table 3.11: Implementation Description: RF17

Feature ID	RF22 (Repository Feature 22)
Name	“Your Preferences” box as part of the user dashboard
Effort Spent	1 Week
Modules Affected / Created	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.

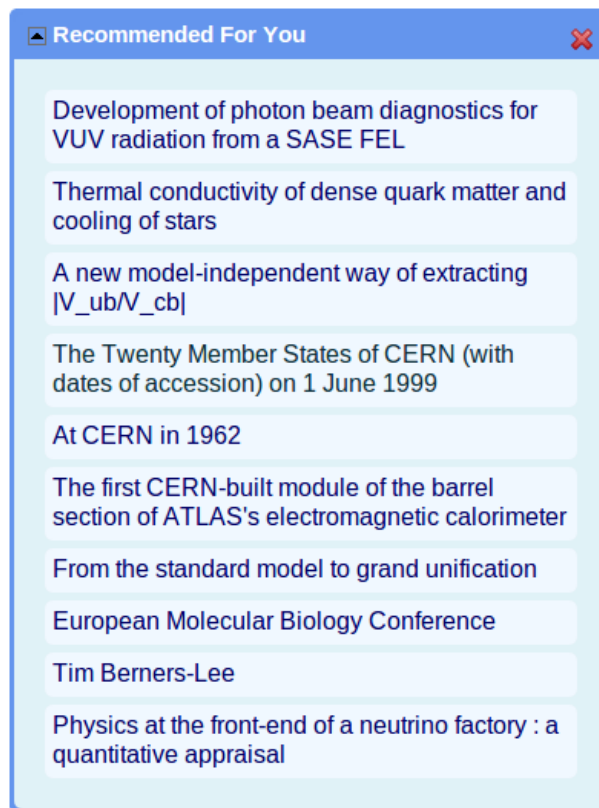


Figure 3.8: RF22 - “Recommended for you” box

Implementation details

- Two new database tables, `query_term` and `user_query_term` have been created. Moreover, `log_query_terms` function has been implemented 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 has been developed. These functions are:
 - `get_unread_records` : Returns the record ids that is not viewed by given user.
 - `get_query_terms` : Returns list of the query terms that provided user searched.
 - `get_recommended_content` : Returns the unread records based on word similarity with the query terms of the provided 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"> • <code>tmpl_recommended_content_box</code> method is added into <code>websession_templates.py</code> to construct the body of the recommendation box. • Some minor changes have been applied in the <code>websession_templates.py</code> and <code>websession_webinterface.py</code> to add new box. 		
<table border="1"> <tr> <td>Implemented By</td> <td>Alper Çınar (SRDC)</td> </tr> </table>	Implemented By	Alper Çınar (SRDC)
Implemented By	Alper Çınar (SRDC)	

Table 3.12: Implementation Description: RF22

Feature ID	RF23 (Repository Feature 23)
Name	The archive stores the comments of blog posts and displays them as part of the blog posts
Effort Spent	3 Days
Modules Affected / Created	BibFormat
Description of the new feature	
<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>	
Implementation details	
<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>	
Implemented By	Raquel Jiménez Encinar (CERN)

Table 3.13: Implementation Description: RF23

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

Implementation details

A new BibFormat element called “BFT_LINKS_MENU” is created for this purpose, 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.

Reference links on this post:

Institutional repository?

*This content is also available
in the archive:*

Institutional repository?

JISCPM answers site

definition contained on Wikipedia

Like

Misc

Figure 3.9: RF24 - Links to other sources menu

Implemented By	Raquel Jiménez Encinar (CERN)
-----------------------	-------------------------------

Table 3.14: Implementation Description: RF24

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

The new BibFormat element “BFT_TAGS” is created which is used in the Bib-Format templates “BlogHTML.bft”, “PostHTML.bft”



Figure 3.10: RF25 - Tags

Implemented By	Raquel Jiménez Encinar (CERN)
-----------------------	-------------------------------

Table 3.15: Implementation Description: RF25

Feature ID	RF26 (Repository Feature 26)
Name	BlogUploader command line to upload, update and delete a list of blogs
Effort Spent	3 Weeks
Modules Affected / Created	WebBlog

Description of the new feature

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.

Implementation details

A new command line tool called “bloguploader” is implemented. The following modes are offered:

- 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 marcxml file that contains all the records to be inserted, deleted or updated by bibupload.

Implemented By	Raquel Jiménez Encinar (CERN)
-----------------------	-------------------------------

Table 3.16: Implementation Description: RF26

Feature ID	RF28 (Repository Feature 28)
Name	The archive displays the author of blog posts and comments
Effort Spent	3 Days
Modules Affected / Created	BibFormat
Description of the new feature	
<p>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.</p>	
Implementation details	
<p>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.</p>	
Implemented By	Raquel Jiménez Encinar (CERN)

Table 3.17: Implementation Description: RF28

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

The screenshot shows a web form titled "Submit a Blog" within a "Blog Submission" context. The form is set against a light green background. At the top, there is a navigation bar with "Blog Submission", "Submit a Blog", "page: 1", and "SUMMARY(2)". The main content area is titled "Submit blog metadata:" and contains the following fields:

- "Blog title:" followed by a text input field.
- "* Blog URL:" followed by a text input field.
- "* Topic:" followed by a dropdown menu showing "- select -".
- "* License:" followed by a dropdown menu showing "- select -".

 At the bottom of the form is a "Finish Submission" button. A footer bar at the very bottom displays "Submission number(1): 1349451030_3138".

Figure 3.11: RF31 - Submit a blog

- 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.

The screenshot shows a web form titled "Modify Blog Information" within a "Blog Submission" context. The form is set against a light green background. At the top, there is a navigation bar with "Blog Submission", "Modify Blog Information", "page: 1", and "SUMMARY(2)". The main content area is titled "Modify blog metadata:" and contains the following fields:

- "Blog URL:" followed by a text input field.
- "* Choose the fields to be modified:" followed by a multi-select dropdown menu with options: "Select:", "Title", "Topic", and "License".

 At the bottom of the form is a "Continue" button. A footer bar at the very bottom displays "Submission number(1): 1349451065_3109".

Figure 3.12: RF31 - Modify a blog

- 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.

Figure 3.13: RF31 - Delete a blog

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

Figure 3.14: RF31 - Delete a post

- 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.

Implemented By	Raquel Jiménez Encinar (CERN)
-----------------------	-------------------------------

Table 3.18: Implementation Description: RF31

Feature ID	RF32 (Repository Feature 32)
Name	Users are able to remove their personal data
Effort Spent	3 Days
Modules Affected / Created	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](#)

Figure 3.15: RF32 - Personal data removal interface

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](#)

Figure 3.16: RF32 - Registration failure message

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](#)

Figure 3.17: RF32 - “Welcome back” message

Implementation details

<ul style="list-style-type: none"> • 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. 	<table border="1"> <tr> <td style="width: 20%;">Implemented By</td> <td>Şenan Postacı(SRDC)</td> </tr> </table>	Implemented By	Şenan Postacı(SRDC)
Implemented By	Şenan Postacı(SRDC)		

Table 3.19: Implementation Description: RF32

Feature ID	RF35 (Repository Feature 35)
Name	The archive displays other blogs that were viewed by people who also viewed the current blog
Effort Spent	2 Days
Modules Affected / Created	BibRank
Description of the new feature	
<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	
<p>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</p>	

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.

Implemented By	Raquel Jiménez Encinar (CERN)
-----------------------	-------------------------------

Table 3.20: Implementation Description: RF35

Feature ID	RF40 (Repository Feature 40)
Name	The archive validates the content received from the spider
Effort Spent	1 Day
Modules Affected / Created	BibUpload, BibIngest
Description of the new feature	
<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 <code>bp_pre_ingestion.py</code> in the pre-ingestion processing described in RF9.</p>	
Implementation details	
<p>See BibIngest module described in RF87.</p>	
Implemented By	Nikolaos Kasioumis (CERN)

Table 3.21: Implementation Description: RF40

Feature ID	RF41 (Repository Feature 41)
Name	The archive detects and eliminates spam content
Effort Spent	3 Weeks
Modules Affected / Created	BlogSpam, BibSched
Description of the new feature	
<p>BlogSpam daemon is scheduled to run periodically and check all repository records to identify spam content. Spam records are flagged with a special MARC tag 911s as spam (1) or not spam (0).</p>	
Implementation details	
<p>BlogSpam is based on URL blacklists such as SpamHaus ^a to identify spam.</p> <p>The operation of the BlogSpam module can be summarized as follows:</p>	

<ol style="list-style-type: none"> 1. BlogSpam iterates over all records 2. For each record, 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. If there is a URL, it checks if the metadata element 911 u (the spam flag) exists. If True, this item is already classified and its skipped. 4. If the record does not have 911 \$u, the spam classifier is checking if it is spam and it is saving the outcome in element 911 u. <p>After the daemon process has been completed, the admin should run bibindex and webcol to see the changes in the records.</p> <p>Configuration file: etc/blogspam/blogspam.cfg</p> <p>Command line execution: sudo -u www-data /opt/invenio/bin/blogspam</p> <p>^ahttp://www.spamhaus.org</p>		
<table border="1"> <tr> <td>Implemented By</td> <td>Vangelis Banos (AUTH)</td> </tr> </table>	Implemented By	Vangelis Banos (AUTH)
Implemented By	Vangelis Banos (AUTH)	

Table 3.22: Implementation Description: RF41

Feature ID	RF45 (Repository Feature 45)
Name	The archive is able to inter-operate with federated search engine dbwiz (SRU Server)
Effort Spent	3 Weeks
Modules Affected / Created	BibFormat, WebSearch, WebStyle
Description of the new feature	
<p>SRU is a standard XML-focused search protocol for Internet search queries. Support for the SRU protocol has been added in Invenio.</p>	
Implementation details	
<p>Any 3rd 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.</p> <p>Example request: http://bf3.itc.auth.gr/sru?version=1.1&operation=searchRetrieve&query=information&maximumRecords=10&recordSchema=dc</p> <p>SRU 1.2 service. parameters:</p> <ul style="list-style-type: none"> • operation → (searchRetrieve, explain, scan, CQL) • version → only 1.2 is supported 	

<ul style="list-style-type: none"> • 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 <p>scan and CQL are not supported yet</p>		
<table border="1"> <tr> <td>Implemented By</td> <td>Apostolos Papadopoulos (ALTEC), Vangelis Banos (AUTH)</td> </tr> </table>	Implemented By	Apostolos Papadopoulos (ALTEC), Vangelis Banos (AUTH)
Implemented By	Apostolos Papadopoulos (ALTEC), Vangelis Banos (AUTH)	

Table 3.23: Implementation Description: RF45

Feature ID	RF47 (Repository Feature 47)
Name	Description of how to cite archived records is presented prominently with each record
Effort Spent	3 Days
Modules Affected / Created	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” (record_creation_date). record_url Retrieved from the original “original_url”

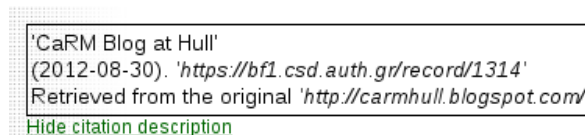


Figure 3.18: RF47 - How to cite box for blogs

- For blog posts:
 - author. “title”. Blog: “blog_title”. (record_creation_date) record_url. Retrieved from the original “original_url”

Paul Anderson. 'A roadmap for low carbon computing'. Blog: 'Notes from the Future'. (2012-08-30). 'https://bf1.csd.auth.gr/record/1338'
Retrieved from the original 'http://notesfromthefuture.jiscinvolve.org/wp/2009/11/24/a-roadmap-for-low-carbon-computing/'
[Hide citation description](#)

Figure 3.19: RF47 - How to cite box for blog posts

- For comments:
author. Blog post: "blog_title". (record_creation_date) record_url. Retrieved from the original "original_url"

Jenny Mitcham. Blog post: 'Engaging with the user community'. (2012-08-30). 'https://bf1.csd.auth.gr/record/1328'
Retrieved from the original 'http://epiccambridge.wordpress.com/2011/06/16/engaging-with-the-user-community/#comment-6'
[Hide citation description](#)

Figure 3.20: RF47 - How to cite box for comments

Implemented By	Raquel Jiménez Encinar (CERN)
-----------------------	-------------------------------

Table 3.24: Implementation Description: RF47

Feature ID	RF48 (Repository Feature 48)
Name	The archive provides the option to translate its content on demand
Effort Spent	2 Weeks
Modules Affected / Created	BibFormat, WebBasket, WebComment, WebMessage, WebStyle
Description of the new feature	
Content of records, messages, reviews, comments and notes on baskets can be translated. The language used by the user determines the language of the translation.	

Thesis / Computing and C

[Show original](#)

Original text:

Systeme Multiagent Pour Les Environnements Riches En Informations (french Text)

Multiagent system for environments rich in information

[Troudi, N.](#) (Laval University)
[Chaib-Draa, B.](#) ; dir.
 1999
 Laval University Laval
 ISBN: 0612382052

Abstract: The growth of the Web is spectacular, with an estimated today has more than 50 million, the nu that only wait to be consulted. A simple calculation shows that devoting even a minute parpage, it would t read all these pages. Using a search strategy is vital. In this context, many research tools have been prop called engines derecherche proved incapable today of providing assistance to users. The main reasons it l nature of the Internet: no central oversight does audeveloppement on the Internet, since anyone who wish provide information is free do, (2) The dynamic nature of information: information not available today and t conversely, (3) The nature of heterogeneous information: Information is available in several formats and se complicating the automatic search of information. Recognizing this, ilsemble important to seek new soluti finding information. (Abstract Shortened by UMI.)

Note: No fulltext; Not held by the library
Notes: MSc: Universite Laval: 1999

Figure 3.21: RF48 - A translated record

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, the user using the platform in English translates the message in German to English.

From: [admin](#)
Subject: Translate
Sent on: 06 Apr 2012, 17:11
Sent to: [admin](#)

[Translate](#)
 Dies ist die Botschaft, werde übersetzt werden soll.

Figure 3.22: RF48 - Translate link in a message

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

From: [admin](#)
Subject: Translate
Sent on: 06 Apr 2012, 17:11
Sent to: [admin](#)

[Show original](#)
 This is the message that will be translated.

Figure 3.23: RF48 - A translated content can be reverted to its original

Implementation details

- Google translate gadget has been utilized for translation task.
- “Translate” link is added over the records, messages, reviews, comments and notes on baskets to translate the content.
- To add “Translate” link, `webstyle_templates.py` , `webmessage_templates.py` , `webcomment_templates.py` , `webbasket_templates.py` and some of the format templates in “bibformat” have been changed.

Implemented By	Şenan Postacı, Alper Çınar (SRDC)
-----------------------	-----------------------------------

Table 3.25: Implementation Description: RF48

Feature ID	RF53 (Repository Feature 53)
Name	The archive respects content licenses and displays useful information about them
Effort Spent	2 Weeks
Modules Affected / Created	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.

Implemented By	Raquel Jiménez, Jaime García (CERN)
-----------------------	-------------------------------------

Table 3.26: Implementation Description: RF53

Feature ID	RF54 (Repository Feature 54)
-------------------	------------------------------

Name	The archive keeps all the different versions of a record
Effort Spent	1 Week
Modules Affected / Created	BibIngest
Description of the new feature	
<p>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.</p>	
Implementation details	
<p>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.</p>	
Implemented By	Nikolaos Kasioumis (CERN)

Table 3.27: Implementation Description: RF54

Feature ID	RF57-58-61 (Repository Feature 57-58-61)
Name	<p>The archive provides a ranking method based on the user rating of content (RF57)</p> <p>A user can rank archived content based on specific users’ content rating (RF58)</p> <p>The archive ranks blogs based on their views and downloads (RF61)</p>
Effort Spent	1 Week
Modules Affected / Created	Bibrank
Description of the new feature	
<p>This feature adds two different ranking method templates to the existing ones: “Number of Record Views” (record_view) and “Average Review Score” (average_score). The admin can add these two methods as ranking methods as it can be seen in the following images:</p>	

Add new rank method

Step 1 - Create new rank method [?](#)

BibRank code:

A unique code that identifies a rank method, is used when running the rank method. The template files include the necessary parameters for the rank method. For more information, please go to the [BibRank guide](#) and read it.

BibRank code

Cfg template [Add rank method](#)

Figure 3.24: RF57-58-61 - New ranking method on adding ranking method admin interface

BibRank Admin Interface

Overview of rank methods [?](#)

[Add new rank method](#)

Code	Translations	Collections	Rank method
average_score	Modify	Modify	Show Details / Modify / Delete
citation	Modify	Modify	Show Details / Modify / Delete
citerank_citation_t	Modify	Modify	Show Details / Modify / Delete
citerank_pagerank_c	Modify	Modify	Show Details / Modify / Delete
citerank_pagerank_t	Modify	Modify	Show Details / Modify / Delete
demo_jif	Modify	Modify	Show Details / Modify / Delete
record_view	Modify	Modify	Show Details / Modify / Delete
wrd	Modify	Modify	Show Details / Modify / Delete

Figure 3.25: RF57-58-61 - Ranking methods on BibRank admin interface

The user can rank the search results with these new ranking methods by selecting one of them in the Advanced Search panel..

Sort by:

Narrow by collection: [Articles & Documents](#)

Ranking methods dropdown:

- or rank by -
- word similarity
- average_score
- citation
- record_view

Figure 3.26: RF57-58-61 - Ranking methods on advanced search panel

“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 if visits occurred within a minute. In the example below, first and second records are viewed 38 and 32 times, respectively.

Sort by: latest first asc. record_view 10 results split by col Display results:

Results overview: Found **176** records in 0.03 second

[Articles & Preprints, 99 records found](#)
[Books & Reports, 48 records found](#)
[Multimedia & Arts, 29 records found](#)

Articles & Preprints

1. **Non-Abelian Flat Directions in a Minimal Su**
(38) 2000-03-P] [TPI-MINN-2000-6] [hep-ph/0002060]
 Recently, by studying exact flat directions of non-Abelian below the stringscale, consists solely of the MSSM spec
 Published in **Mod. Phys. Lett. A: 15 (2000) pp. 1191-12**
Fulltext: [PDF](#) [PS.GZ](#);
[Detailed record](#) - [Similar records](#)
2. **Leptogenesis with Majorana neutrinos / Pas**
(32) I review the origin of the lepton asymmetry which is con
Fulltext: [PDF](#) [PS.GZ](#);
[Detailed record](#) - [Similar records](#) - [1 comment](#) - [2 reviews](#)
3. **Development of photon beam diagnostic f**

Figure 3.27: RF57-58-61 - Ranking by record view number

“Average Review Score” calculates the average scores of the records. In the example below, records have average score of 4.5, 3.0 and 2.5 respectively.

latest first asc. average_score 10 results split by col

Results overview: Found **176** records in 0.01 second

[Articles & Preprints, 99 records found](#)
[Books & Reports, 48 records found](#)
[Multimedia & Arts, 29 records found](#)

Articles & Preprints

1. **Leptogenesis with Majorana neutrinos / Pa**
(4.5) I review the origin of the lepton asymmetry which is con
Fulltext: [PDF](#) [PS.GZ](#);
[Detailed record](#) - [Similar records](#) - [1 comment](#) - [2 review](#)
2. **Study of a dE/dx measurement and the gas**
(3.0) Preprint-95-196] [TUAT-HEP-96-1] [DPNU-96-04] [SCAI
 Published in **Nucl. Instrum. Methods Phys. Res., A :3**
Fulltext: SCAN-9605071 - [TIF](#);
[Detailed record](#) - [Similar records](#) - [1 review](#)
3. **Restrictions on Gauge Groups in Noncomm**
(2.5) We show that the gauge groups SU(N), SO(N) and Sp(N)
 Published in **Phys. Lett., B :482 2000 417-419**

Figure 3.28: RF57-58-61 - Ranking by average score

Implementation details

- 2 new ranking templates have been implemented: **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 counted only once.

<ul style="list-style-type: none"> • 4 new functions have been implemented in bibrank_tag_based_indexer.py : <ul style="list-style-type: none"> – record_view : executes bibrank_engine method for record_view ranking method – record_view_exec : Ranks total number of record visits without checking the user ip – average_score : executes bibrank_engine method for average_score ranking method – average_score_exec : Ranks average review score for records • 2 new files have been created: bibrank_record_view_indexer.py and bibrank_average_score_indexer.py <ul style="list-style-type: none"> – bibrank_record_view_indexer contains the functions that are used for indexing visit counts of each record. – bibrank_average_score_indexer contains the function that are used for indexing average review score of each record. 	<table border="1"> <tr> <td style="width: 20%;">Implemented By</td> <td>Şenan Postacı, Alper Çınar (SRDC)</td> </tr> </table>	Implemented By	Şenan Postacı, Alper Çınar (SRDC)
Implemented By	Şenan Postacı, Alper Çınar (SRDC)		

Table 3.28: Implementation Description: RF57-58-61

Feature ID	RF59 (Repository Feature 59)										
Name	Export data using XML (METS, MARC)										
Effort Spent	2 Weeks										
Modules Affected / Created	BibUpload, MiscUtil										
Description of the new feature											
<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 & Transmission Standard).</p>											
Implementation details											
<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>											
<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>column name</th> <th>value</th> </tr> </thead> <tbody> <tr> <td>name</td> <td>METS</td> </tr> <tr> <td>code</td> <td>xmets</td> </tr> <tr> <td>description</td> <td>Metadata Encoding & Transmission Standard</td> </tr> <tr> <td>content_type</td> <td>text/xml</td> </tr> </tbody> </table>	column name	value	name	METS	code	xmets	description	Metadata Encoding & Transmission Standard	content_type	text/xml	
column name	value										
name	METS										
code	xmets										
description	Metadata Encoding & Transmission Standard										
content_type	text/xml										
<p>This is how this new option looks like on the main search page:</p>											

➔ **Export as [BibTeX](#), [MARC](#), [MARCXML](#), [DC](#), [EndNote](#), [NLM](#), [RefWorks](#), [PDF](#), [JPEG](#)**

Figure 3.32: RF62 - PDF and JPEG as export options

Pdfs are created in latex template. First the record in html form (with its css style) is converted to latex source. The latex source is later, with XeLaTeX engine, converted to pdf file.

Higher dimensional geometries related to fuzzy odd-dimensional spheres

Ramgoolam, S

Jul, 2002

We study $SO(m)$ covariant Matrix realizations of $\sum_{i=1}^m X_i^2 = 1$ for even m as candidate fuzzy odd spheres following hep-th/0101001. As for the fuzzy four sphere, these Matrix algebras contain more degrees of freedom than the sphere itself and the full set of variables has a geometrical description in terms of a higher dimensional coset. The fuzzy S^{2k-1} is related to a higher dimensional coset $\frac{SO(2k)}{U(1) \times U(k-1)}$. These cosets are bundles where base and fibre are hermitian symmetric spaces. The detailed form of the generators and relations for the Matrix algebras related to the fuzzy three-spheres suggests Matrix actions which admit the fuzzy spheres as solutions. These Matrix actions are compared with the BFSS, IKKT and BMN Matrix models as well as some others. The geometry and combinatorics of fuzzy odd spheres lead to some remarks on the transverse five-brane problem of Matrix theories and the exotic scaling of the entropy of 5-branes with the brane number.

Figure 3.33: RF62 - Jpeg of a record

Jpegs of the records are created by taking snapshots of the records.

Implementation details

- A new bibformat element namely **bfe_latex_main_template.py** responsible for pdf has been created. The header section(title, authors and date) of the latex code is constructed in this module.
- Record fields with tag **'520__a'** and **'520__b'** are used as record body. The record body is in html form(the records in blogforever.cern.ch demo site are also in html form).
- To parse html formatted data and convert it to latex template, **bibformat_-pdf_with_latex_template.py** and **bibformat_pdf_with_latex_template_-config.py** modules have been implemented.
- The **bibformat_pdf_with_latex_template_config.py** module contains configuration parameters for html to pdf conversion. Most of the parameters are related with latex representation of html tags and styles. Each html tag, css rule or special html character has possible latex representation as value. An example of this:

```
CFG_BIBFORMAT_LATEX_REPRESENTATION_OF_HTML_TAGS = {
```

```

'h1'      : {'end': '}\n', 'start': '\n\\section{\n'},
'h2'      : {'end': '}\n', 'start': '\n\\subsection{\n'},
'h3'      : {'end': '}\n', 'start': '\n\\subsubsection{\n'}
.
}
CFG_BIBFORMAT_LATEX_REPRESENTATION_OF_CSS_RULES = {
.
'text-decoration': {
    'underline': {
        'start': r'\underline{',
        'end': '}'
    },
    'overline': {
        'start': r'\overline{',
        'end': '}$'
    },
    'line-through': {
        'start': r'\sout{',
        'end': '}'
    }
}
.
}

```

The config file also contains regular expressions to parse html and css data and optional parameters.

To successfully run this feature, there are some configuration parameters related with auxiliary tools such as XeLaTeX or path to directory where pdfs and jpegs will be saved. They have default values, however, it would be a good idea to check whether these configuration parameters coincide with your system settings. All parameters are described with comments.

There are also some configuration parameters affecting run time behaviours of this feature. For example, **CFG_BIBFORMAT_CSS_FILES** consists of paths to the css files to be used for the record page.

- The **bibformat_pdf_with_latex_template.py** module is like an engine that performs the conversion. **PdfWithLatexTemplateHtmlParser** class extending python HTMLParser class parses the html data, finds start and end tags and calls applicable converter function. The **LatexConverter** class consists of converter functions. Most of the html tags are simply mapped from the dictionaries in config file. However, some of them, e.g., table, tr, td, img, font need extra effort.
- The style of the html based record is also taken into consideration if css rules are provided.
 - Css style can be provided in two ways. One way is that the path to the css files can be added (**CFG_BIBFORMAT_CSS_FILES**) list under config file. The other method is that the css rules, in string type, can be provided to **CssParser** class.
 - Rules are parsed and kept as dictionary. For advanced style process, possible selectors, e.g., div p, #id, .classname #id, tag > #id .classname, are

constructed first and then if there are any rules defined for a selector, it is applied to latex code.

- At present, basic and most frequently used style rules are handled. However, extra effort is needed for remaining rules and more complex style selectors.
- After html to latex conversion is completed, a pdf file is created with XeLaTeX engine. For this operation LaTeX TeX Live distribution must be installed. To install TeX Live **install-texlive** command is added into Makefile. After the "PDF" link is clicked, under `<yourwebdir>/export/`, a temporary directory for the user is created. The directory name is the user's session key. When different users simultaneously use this feature, to prevent conflicts this mechanism is applied. Pdf file(s) are generated under users' temporary directories. Later, the pdf content is written into **req** response object and the `content_type` of the req response object is set to `application/pdf`.
- For exporting a record as jpeg, its snapshot is taken and then the snapshot is provided to the user. To take the snapshot a record page is created without any header or footer. The url of the "snapshot page" is provided to the "phantomjs" webkit tool which is used for taking the screenshot. To install "phantomjs" tool, one of the commands **install-phantomjs-64bits** or **install-phantomjs-32bits** in Makefile can be used.
- "simplr" (simple record page) and "jpeg" output formats are created for this operation. The former one is responsible for creating the "snapshot" page which includes only record content. When a request is made for the latter one, under **bibformat.py** , **create_jpeg** function is called which runs "phantomjs" to create the image of the record.
- The "phantomjs" tries to connect the site from outside without any login information of the user. Therefore, it is impossible for it to access the restricted records even if the user has permission to access. To enable "phantomjs" access to the restricted collections, the user's `session_key` is provided in the url. For example:
https://your_site_url/record/105/export/recordcontent?ln=en&session=a407af957cde04031a99793a21fe643f
 If the `session_key` is validated and verified that the user has permission to access the corresponding record, then the "snapshot page" is created and the snapshot is taken. **recordcontent** method has been added in class **WebInterfaceRecordExport** in **websearch_webinterface.py** to do so.
- If the number of records to be converted to pdf/jpeg is more than one, then each pdf/jpeg is generated and all of them are zipped. The users are able to download this zip.
- All output files are removed after they are provided to the user. Therefore, for concurrent requests, the files may be deleted before they are provided to the request owners. To prevent this, jpegs and pdfs are created in a temporary directory which is unique for the user and after the work is done this directory is removed completely.

Implemented By	Şenan Postacı(SRDC)
-----------------------	---------------------

Table 3.30: Implementation Description: RF62

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

Description of the new feature

This feature introduces three external login systems: OpenID, OAuth1 and OAuth2.

Users can login with third party sites like facebook, google, twitter etc. as well as the internal login system.

This feature lets the user choose the third party site in the login page.



Figure 3.34: RF64 - External login providers

When the user selects one of these providers, s/he is redirected to login page of the 3rd party site. After logged in successfully, the user redirected back to the internal system.

Adding an OpenID provider (Verisign Example)

1. Add configurations of verisign into **CFG_OPENID_CONFIGURATIONS** in **access_control_config.py** , **{0}** will be replaced by the input of the login form.

```
CFG_OPENID_CONFIGURATIONS = {
    ...,
    'verisign' : {
        'identifier' : '0.pip.verisignlabs.com' ,
        'trust_email' : False
    },
    ...
}
```

Since verisign lets the users change their email address, set **'trust_email'** **False** (or do not add **'trust_email'** key)

2. Add **'verisign'** to **CFG_OPENID_PROVIDERS** to start to use verisign.

```
CFG_OPENID_PROVIDERS = [
    ...,
```

```

    'verisign' ,
    ...
]

```

- To edit the label of the form displayed in the login page, add

```

CFG_EXTERNAL_LOGIN_FORM_LABELS = {
    ...,
    'verisign' : 'Your VeriSign username' ,
    ...,
}

```

otherwise it will be `verisign username` .

- Add two images `verisign_icon_24.png` ✓ and `verisign_icon_48.png` ✓ -which are 24px*24px and 48px*48px respectively- into image folder `.../invenio/var/www/img/`

Adding an OAuth2 provider (FourSquare Example):

- Add your application to provider site:

Figure 3.35: RF64 - Adding an application to FourSquare

Application web site and callback url (redirect url) should be `CFG_SITE_SECURE_URL` and `CFG_SITE_SECURE_URL/youraccount/login?login_method=oauth2&provider=foursquare` respectively. The value of the `provider` parameter in the redirect URI should be same as the key added into `CFG_OAUTH2_CONFIGURATIONS` in `access_control-config.py` .

The provider will give you Consumer Key and Consumer Secret. (Here Client ID and Client Secret)

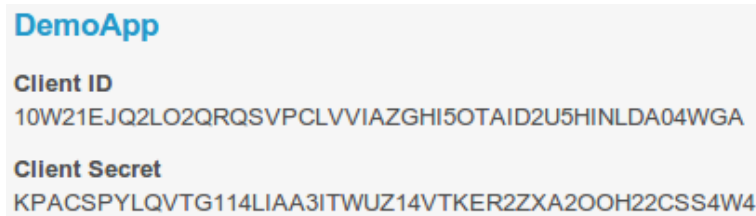


Figure 3.36: RF64 - Client ID and Secret obtained from FourSquare

2. Find the `authorize_url` and `access_token_url` of the provider. For foursquare they are:
 - `access_token_url`: `https://foursquare.com/oauth2/access_token`
 - `authorize_url`: `https://foursquare.com/oauth2/authorize`
3. Find the url to get the user information. (It is generally an url which needs access token in parameters) For foursquare it is: `https://api.foursquare.com/v2/users/self?oauth_token={access_token}`
4. Add the configurations to the **CFG_OAUTH2_CONFIGURATIONS** in **access_control_config.py** and enable the provider by adding the **CFG_OAUTH2_PROVIDERS** :

```
CFG_OAUTH2_CONFIGURATIONS = {
    ...,
    'foursquare' : {
        'consumer_key' :
            '10W21EJQ2LO2QRQSVPCLVVIAZGHI50TAID2U5HINLDA04WGA' ,
        'consumer_secret' :
            'KPACSPYLQVTG114LIAA3ITWUZ14VTKER2ZXA2OOH22CSS4W4' ,
        'access_token_url' :
            'https://foursquare.com/oauth2/access_token' ,
        'authorize_url' :
            'https://foursquare.com/oauth2/authorize' ,
        'request_url' :
            'https://api.foursquare.com/v2/users'
            '/self?oauth_token={access_token}' ,
        'debug' : 1
    },
    ...
}

CFG_OAUTH2_PROVIDERS = [
    ...,
    'foursquare' ,
    ...
]
```

5. Go to login page and login with foursquare. Since it is not configured yet, only the json object we get from provider will be displayed after logging in. Now we need to find the email and the id of the user from that json object:

```

{
  ...,
  'response' : {
    'user' : {
      ...,
      'id' : '30362394' ,
      ...,
      'contact' : {
        ...,
        'email' : 'some-email' ,
        ...
      },
      ...
    }
  },
  ...
}

```

Now we can complete the configuration, delete the **'debug'** key and insert the path of the e-mail and ID. (ID is required)

```

CFG_OAUTH2_CONFIGURATIONS = {
  ...,
  'foursquare' : {
    'consumer_key' :
      '10W21EJQ2LO2QRQSVPCLVVIAZGHI50TAID2U5HINLDA04WGA' ,
    'consumer_secret' :
      'KPACSPYLQVTG114LIAA3ITWUZ14VTKER2ZXA200H22CSS4W4' ,
    'access_token_url' :
      'https://foursquare.com/oauth2/access_token' ,
    'authorize_url' :
      'https://foursquare.com/oauth2/authorize' ,
    'request_url' :
      'https://api.foursquare.com/v2/users'
      '/self?oauth_token={access_token}' ,
    'id' : [ 'response' , 'user' , 'id' ],
    'email' : [ 'response' , 'user' , 'contact' ,
      'email' ]
  },
  ...
}

```

Note that, since the email is reached by `jsonobject['response']['user']['contact']['email']`, the path of the e-mail should be specified in configuration as `'email' : ['response' , 'user' , 'contact' , 'email']` and id is similar.

6. Add **foursquare_icon_24.png**  and **foursquare_icon_48.png**  into the image folder (`.../invenio/var/www/img/`).

Some providers need extra arguments when authorizing user. These arguments should be specified in **'authorize_parameters'** key of the configuration.

For example, facebook needs scope parameter in authorize_url to get the email. Then we need to set:

```
CFG_OAUTH2_CONFIGURATIONS = {
    ...,
    'facebook' : {
        ...,
        'authorize_parameters' : {
            'scope' : 'email'
        },
    },
    ...
}
```

Adding an OAuth1 provider:

Adding OAuth1 provider is very similar to add OAuth2 provider. The differences are:

- OAuth1 provider needs request_token_url and additional parameters for request_url (whose keys are **'request_token_url'** and **'request_parameters'** respectively)
- Callback url should be in the form of **CFG_SITE_SECURE_URL** /youraccount/login?login_method=oauth1&provider=providername

Implementation details

- 3 new external authentication modules added: **external_authentication_openid.py**, **external_authentication_oauth1.py** and **external_authentication_oauth2.py**. Unlike the other external authentication methods, these methods have to fetch external ID of the user. E-mail and nickname are optional. If the email cannot be fetched, the system generates a unique email for the user.
- This feature depends on 2 packages: **python-openid** and **rauth** which can be installed by running **make install-openid-package** and **make install-oauth-package**, respectively.
- To enable/disable these authentication methods, **CFG_OPENID_AUTHENTICATION**, **CFG_OAUTH1_AUTHENTICATION** and **CFG_OAUTH2_AUTHENTICATION** parameters have been added into **invenio.conf**
- New module **containerutils.py**, which contains the functions for basic containers such as dict, list and string, created.
- The function **remove_temporary_emails**, which deletes the auto-generated emails from an email list, has been added into **mailutils.py**.
- The table **oauth1_storage** has been inserted into the database to store request tokens.

- Unique key of the table `userEXT` had been removed since a user may have multiple external accounts.
- Some variables have been added into access control config:
 - `CFG_TEMP_EMAIL_ADDRESS` : Template of the temporary email address for the users logged in with an external provider which does not supply their email address.
 - `CFG_OPENID_PROVIDERS` : The list of the activated OpenID providers.
 - `CFG_OAUTH1_PROVIDERS` : The list of the activated OAuth1 providers.
 - `CFG_OAUTH2_PROVIDERS` : The list of the activated OAuth2 providers.
 - `CFG_OPENID_CONFIGURATIONS` : A dictionary to configure the OpenID providers. Its keys are the name of the providers and values are also dictionaries which has at most 2 keys:
 - * `'identifier'` (required): The identifier url of the OpenID provider. The user name area should be replaced by `{0}` .
Example: `openid.aol.com/{0}`
 - * `trust_email` (optional): Set it `False` (or do not add it to keys) for the providers which let the user change his/her email during login process. Otherwise, set it `True` .
 - `CFG_OAUTH1_CONFIGURATIONS` : A dictionary to configure the OAuth1 providers. Its keys are the name of the providers and values are also dictionaries which has at most 12 keys (6 required, 6 optional):
 - * `consumer_key` (required): The **Consumer Key (Client ID, Application ID)** obtained from the OAuth1 provider.
 - * `consumer_secret` (required): The **Consumer Key (Client Secret, Application Secret)** obtained from the OAuth1 provider.
 - * `authorize_url` (required): The url to redirect the user to authorization page.
 - * `authorize_parameters` (optional): A dictionary of additional parameters of the authorization. Parameter name and the value corresponds to key and the value of the dictionary, respectively. (ie `url?param=value`) corresponds to `{ 'param' : 'value' }`
 - * `request_token_url` (required): A url to get the **request token** and **request token secret**.
 - * `access_token_url` (required): A url to exchange the request token with the access token.
 - * `request_url` (optional): A url to get the information of the user.
 - * `request_parameters` (optional): The dictionary of the additional parameters of the `request_url` . (similar to authorize parameters)
 - * `id` (required): The path of the external ID of the user in the returned JSON object from provider. (usage: if the ID is gathered by `jsonobject['response']['user']['id']`, this value should be `['response', 'user', 'id']`)
 - * `email` (optional): The path of the e-mail of the user in the returned JSON object from provider. Usage is similar to `id` .
 - * `nickname` (optional): The path of the nickname of the user in the returned json object from provider. Usage is similar to `id` .

<ul style="list-style-type: none"> * debug (optional): Debug mode of the OAuth1 provider. 1 for activate, 0 for deactivate. If it is activated, it does not log in, just displays the json object returned from provider after login screen to figure out where the ID, e-mail and nickname is. – CFG_OAUTH2_CONFIGURATIONS : A dictionary to configure the OAuth2 providers. Its keys are the name of the providers and values are also dictionaries which has at most 10 keys (6 required, 4 optional): <ul style="list-style-type: none"> * consumer_key , consumer_secret , authorize_url , authorize_parameters , id , email , nickname and debug are the same as CFG_OAUTH1_CONFIGURATIONS * access_token_url (required): A url to get the access token. * request_url (optional): A url to get the user information. Access token (Oauth token) area should be replaced by {access_token} (ie. url?access_token={access_token}) • Suitable warning messages are inserted into CFG_WEBACCESS_WARNING_MSGS in access_control_config.py • The function external_user_warning has been added into webaccount.py , which returns the warning if the email of the user is auto-generated. • The variable warning_list in the function perform_display_account in webaccount.py has been extended with warning of the external user. • Variables has been added into websession_config.py : <ul style="list-style-type: none"> – CFG_EXTERNAL_LOGIN_LARGE : The list of provider names. Decides which login buttons will be displayed 48x48px in the login page. The order of the list also changes the order of the login buttons. – CFG_EXTERNAL_LOGIN_BUTTON_ORDER : The list deciding the order of the login buttons in the login page. The activated but un-ordered buttons will be displayed alphabetically after the ordered ones. – CFG_EXTERNAL_LOGIN_FORM_LABELS : The dictionary whose keys are OpenID providers which needs username for their identifier urls and values are the label of the input in login form. • The functions tmpl_external_login_button , tmpl_external_login_form and tmpl_external_login_panel are added into websession_templates.py to construct the external login panel in login page. • openid , oauth1 , oauth2 methods have been added into websession_webinterface.py . These functions redirects the user authorization url or displays messages in case of errors. • loginUser function in webuser.py is regulated to accommodate OpenID/OAuth authentication. 	<table border="1" style="width: 100%;"> <tr> <td style="width: 25%;">Implemented By</td> <td>Alper Çınar (SRDC)</td> </tr> </table>	Implemented By	Alper Çınar (SRDC)
Implemented By	Alper Çınar (SRDC)		

Table 3.31: Implementation Description: RF64

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

Description of the new feature	
<p>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.</p>	
Implementation details	
<p>The script <code>spider_repository_communication.py</code> 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.</p>	
Implemented By	Raquel Jiménez, Jaime García (CERN)

Table 3.32: Implementation Description: RF67

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

Description of the new feature

This feature introduces “premium access to collections” system.

Admin may restrict collections for some cost for finite/infinite time interval.

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

Admin may add premium packages for collections:

Add New Premium Package

Name:

Details:

Duration:

Price: .

Collections:

✘
 Select at least 1 collection.

Figure 3.37: RF70 - Adding new premium package panel

Current premium packages can be monitored, edited or deleted:

Edit Packages						
ID	Name	Details	Duration	Price	Access	
1	Articles and Books	View Articles and Books for 3 days!	3 Day	5.00 EUR	Books Articles	
2	Articles - 1	View articles for a month!	1 Month	19.00 EUR	Articles	
3	Articles - 2	How about viewing articles forever?	Unlimited	239.00 EUR	Articles	
4	Articles - 3	View articles for 3 hours!	3 Hour	2.00 EUR	Articles	
5	Books - 1	Read books for a whole year!	1 Year	99.00 EUR	Books	
6	Books - 2	I bet you can't read a book in 1 week!	1 Week	9.00 EUR	Books	

[Add New Premium Package](#)

Figure 3.38: RF70 - Monitoring premium packages in admin panel

- 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.

After the premium packages are added, collections become restricted.

Articles & Preprints

Search 0 records for:

 any field

[Search Tips](#) :: [Advanced Search](#)

Narrow by collection:

- [Articles](#) (0) [restricted]
- [Preprints](#) (0)

Search also:

- [CERN Indico](#)
- [CiteSeer](#)
- [INSPIRE](#)
- [KISS Books/Journals](#)
- [KISS Preprints](#)
- [Scirus](#)

Figure 3.39: RF70 - A restricted collection that requires premium package to display

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

Articles

Search 0 records for:

 any field

[Search Tips](#) :: [Advanced Search](#)

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

Figure 3.40: RF70 - Warning message indicates that the collection is restricted

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

Authorization failure

Here is the list of packages to access Articles

Package Name	Details	Duration	Price	Access to Collections
<input type="radio"/> Articles - 1	View Articles for an hour!	1 Hour	EUR 0.99	Articles
<input type="radio"/> Articles - 2	View Articles for a day!	1 Day	EUR 1.99	Articles
<input type="radio"/> Articles - 3	View articles for 3 hours!	3 Hours	EUR 2.00	Articles
<input type="radio"/> Articles and Books	View articles and books for 3 days!	3 Days	EUR 4.98	Books Articles
<input type="radio"/> Articles and Preprints	View articles and preprints for 8 hours!	8 Hours	EUR 7.77	Preprints Articles
<input type="radio"/> All-in-One	View all restricted collections for a year!	1 Year	EUR 125.97	Preprints Books Theses Reports Articles Pictures Poetry Videos



Figure 3.41: RF70 - Available premium packages to display current collection

User selects a suitable package and payment method:

1) Credit card

Upgrade Account

Premium Package Details

Name: Articles - 3
 Details: View articles for 3 hours!
 Duration: 3 Hours
 Price: 2.00 EUR
 Access to Collections: Articles

Credit Card Information

Name on Card:
 Card Number:
 Expiration: /
 Security Code:

Personal Information

First Name:
 Last Name:

Address Information

Street:
 City:
 State / Province:
 Zip / Postal Code:
 Country:

Figure 3.42: RF70 - A form to purchase a premium package with credit card

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

This transaction cannot be processed. Please enter a valid credit card expiration date.

Premium Package Details

Name: Articles - 3
 Details: View articles for 3 hours!
 Duration: 3 Hours
 Price: 2.00 EUR
 Access to Collections: Articles

Figure 3.43: RF70 - The error message states that the credit card information is wrong

Otherwise, a confirmation page is loaded.

Upgrade Account

You successfully joined the premium groups or extended your membership. Here is the list of your premium groups.

Group Name	Description	Expire	Extend
6 - Articles Premium Group	Premium users who can view Articles	2012-09-18 16:55:04	

Figure 3.44: RF70 - Confirmation page that also displays the premium packages the user have

2) PayPal Express Checkout

SRDC's Document Server

Your order summary

Descriptions	Amount
Books - 2 Item description: I bet you can't re... Item price: €9.00 Quantity: 1	€9.00
Item total	€9.00
Total €9.00 EUR	

Review your information

[Continue](#)

Payment methods [Change](#)

PayPal Balance \$12.60 USD

PayPal Conversion Rate as of Sep 18, 2012: 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 SRDC's Document Server.

[Cancel and return to SRDC's Document Server.](#)

Figure 3.45: RF70 - PayPal express checkout screen

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.

Upgrade Account

Package Name	Details	Duration	Price
Books - 2	I bet you can't read a book in 1 week!	1 Week	9.00 EUR
			Total: 9.00 EUR

The safer, easier way to pay

Figure 3.46: RF70 - PayPal transaction confirmation page

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

Upgrade Account

You successfully joined the premium groups or extended your membership. Here is the list of your premium groups.

Group Name	Description	Expire	Extend
6 - Articles Premium Group	Premium users who can view Articles	2012-09-18 16:55:04	
3 - Books Premium Group	Premium users who can view Books	2012-09-25 14:45:05	

Figure 3.47: RF70 - Confirmation page that also displays the premium packages the user have

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

Your Account

Your Account

You are logged in as jekyll. You may want to a) [logout](#); b) edit your [account settings](#).

Your Messages

You have **0** new messages out of **0** messages.

Your Baskets

You have 0 personal baskets and are subscribed to 0 group baskets and **0** other users public basket

Your Alert Searches

You own the following alerts::

Your Searches

You have made 0 queries. A [detailed list](#) is available with a possibility to (a) view search results an

Your Groups

You can consult the list of [your groups](#) you are administering or are a member of.

Your Administrative Activities

You are enabled to the following roles: *claimpaperusers, thesesviewer, basketusers, alertusers, loan messageusers, statisticsusers, groupusers, premium_collection6_viewer.*

Here are some interesting web admin links for you:

[Run BibSword Client](#)

For more admin-level activities, see the complete [Admin Area](#).

Your Premium Group Memberships

Group Name	Description	Expire	Extend
6 - Articles Premium Group	Premium users who can view Articles	2012-09-18 16:55:04	
3 - Books Premium Group	Premium users who can view Books	2012-09-25 14:45:05	

Figure 3.48: RF70 - “Your Account” page displaying current premium groups the user joined

A user may extend his/her premium group membership by clicking button.

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

Payment History						
Total Number of Transactions: 2						
Total Revenue: 11.00 EUR						
Transaction Time	User ID	User E-Mail	Package ID	Price	Payment Method	
2012-09-18 14:45:05	2	jekyll@cds.cern.ch	6	9.00 EUR	paypal	
2012-09-18 13:55:04	2	jekyll@cds.cern.ch	4	2.00 EUR	Credit Card	

Figure 3.49: RF70 - The admin panel displaying payment history

Admin may give a premium package to a user via the “Upgrade User” form.

User ID	User E-Mail	User Nickname	Collection	Expiration Time
2	jekyll@cds.cern.ch	jekyll	Books	2012-09-25 14:45:05
2	jekyll@cds.cern.ch	jekyll	Articles	2012-09-18 16:55:04

Figure 3.50: RF70 - The admin panel displaying current users joined premium groups

The admin can disable paying with credit card, paypal express checkout or whole premium service by **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 administrator disables the premium service, all the collections become unrestricted. If s/he enables it back, they will become restricted 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**, all of the premium collections become open collections and the information about these premium collections are saved on **accROLE_accACTION_accARGUMENT_inactive** table in the database. On the contrary, If this variable is changed from **0** to **1**, all of the saved information about premium collections are backed up, premium collections become restricted again.
 - **CFG_TEST_PREMIUM_SERVICE** : **1** for use test servers of the payment gateways, otherwise **0**.
 - **CFG_CREDIT_CARD_PAYMENT_GATEWAY** : The payment gateway used for buying 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.
- **Callback functionality is added for updating **config.py** :**
 - Some modifications are occurred in **cli_cmd_update_config.py** function in **invenioconf.py**. This function calls the “callback functions” specified in **CONFIG_PY_CALLBACK** in **invenioconf_callback.py** after updating **config.py**.
 - **CONFIG_PY_CALLBACK** is a dictionary that takes variable names as keys and callback functions as values. Callback functions should get

2 arguments: **old_value** and **new_value**, the prototype of the callback functions is following:

```
sample_callback_function(old_value=None, new_value=None)
```

- **5 new tables added into database:**

- **premium** : Keeps the information (name, details, duration, price and display order) about premium packages.
- **hstPAYMENT** : Keeps the payment history.
- **collection_usergroup_role** : keeps the collection - usergroup - role mapping.
- **premium_collection** : keeps the premium package - collection mapping.
- **accROLE_accACTION_accARGUMENT_inactive** : keeps the the premium roles when the premium service is inactive.
- **expire** column is added into **user_usergroup** table. This column keeps the expiration date of the group membership of a user. Expiration dates of the premium group memberships are calculated when the user purchased a premium package. The expiration dates of any other group memberships are **9999-12-31 23:59:59** as default.
- Whether a collection needs a premium group membership to be accessed is checked when displaying corresponding collection. **acc_authorize_action** function in **access_control_engine.py** is modified to accommodate billing system. It can be viewed if the premium service is enabled and if there is a premium group 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 have premium membership can be displayed in the admin page. To achieve these functionalities, some functions have been implemented in **webaccessadmin.lib.py** and **webaccessadmin.py** . Also **webaccessadmin.js** and **webaccessadmin.min.js** have been added to make the interface more useful.
- The new module **WebPayment** has been 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 new premium package and arranges the roles and authorizations for the corresponding collections.
 - * **edit_premium_package** : Edits an existing premium package, arranges the roles and authorizations about the corresponding collections. If a collection loses its premium packages after editing, that collection becomes unrestricted.
 - * **fix_premium_table_parameters** : Arranges the parameters of **add_new_premium_package** and **edit_premium_package** functions in case of any problematic parameter.
 - * **create_new_premium_group** : Creates a premium group to access the restricted collection. If a group that has the same name as the premium group is created by a user, the user-created group is renamed.
 - * **add_role_and_authorization** : Adds/edits the roles and authorizations to access premium collections.

- * **display_possible_packages** : Displays the packages that requires to access the collection given in the arguments.
- * **get_package_collection_map** : Returns the map of which premium packages allow to access which collections.
- * **upgrade_user** : Gives a user a premium package stated in the arguments.
- * **delete_premium_package** : Deletes the given premium package. If a collection loses its premium packages after deleting, that collection becomes unrestricted.
- * **cfg_premium_service_callback** : callback function of the **CFG_PREMIUM_SERVICE** variable. If the premium services are disabled, it makes all the collections unrestricted. If the premium services are enabled back, it makes corresponding collections restricted again.
- **webpayment_dblayer.py** contains the database related functions of **WebPayment** module.
- **webpayment_base.py** contains required classes to implement payment gateways:
 - * **CreditCard** : This class is inherited from **dict** class. It just ensures that the dictionary contains card number, name on the card, expiration date and security code.
 - * **PaymentGatewayResponse** : This class is also 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. Its type is:


```
[{ 'legend' : str , 'inputs' : [{ 'title' : str , 'name' : str , 'type' : str }]}]
```

 - **legend** : The title of the input set
 - **title** : The title of the input
 - **name** : The name of the input
 - **type** : can be **'text'** or **'country-select'** , to construct **Selecting Country** input, ISO codes of the countries are added into **websession.config.py** as **COUNTRY_ISO_CODES** variable.

For example, in ogone there is no additional inputs and form in the web interface is following:

Credit Card Information	
Name on Card:	<input type="text"/>
Card Number:	<input type="text"/>
Expiration:	<input type="text" value="1"/> / <input type="text" value="2012"/>
Security Code:	<input type="text"/>

Figure 3.51: RF70 - Ogone purchase with credit card from

However, in PayPal it is specified as:

```
_additional_inputs = [{
  'legend' : 'Personal Information' ,
  'inputs' : [{
    'title' : 'First Name' ,
    'name' : 'fname' ,
    'type' : 'text' }, {
    'title' : 'Last Name' ,
    'name' : 'lname' ,
    'type' : 'text' }]
}, {
  'legend' : 'Address Information' ,
  'inputs' : [{
    'title' : 'Street' ,
    'name' : 'street' ,
    'type' : 'text' }, {
    'title' : 'City' ,
    'name' : 'city' ,
    'type' : 'text' }, {
    'title' : 'State / Province' ,
    'name' : 'state' ,
    'type' : 'text' }, {
    'title' : 'Zip / Postal Code' ,
    'name' : 'zip' ,
    'type' : 'text' }, {
    'title' : 'Country' ,
    'name' : 'country' ,
    'type' : 'country-list' }]
}]
```

and the credit card form is following:

Credit Card Information	
Name on Card:	<input type="text"/>
Card Number:	<input type="text"/>
Expiration:	<input type="text" value="1"/> / <input type="text" value="2012"/>
Security Code:	<input type="text"/>

Personal Information	
First Name	<input type="text"/>
Last Name	<input type="text"/>

Address Information	
Street	<input type="text"/>
City	<input type="text"/>
State / Province	<input type="text"/>
Zip / Postal Code	<input type="text"/>
Country	<input type="text" value="AALAND ISLANDS"/>

Figure 3.52: RF70 - PayPal purchase with credit card from

- **name** : the name of the payment gateway
- **_accept_types** : list of the credit card types that accepted. It may contains **PaymentGateway . VISA** , **PaymentGate-**

way . **MASTERCARD** , **PaymentGateway** . **DISCOVER** , **PaymentGateway** . **AMERICANEXPRESS** and **PaymentGateway** . **MAESTRO** constants.

- If the payment gateway is used for buying with credit card, **process** method should be **overridden**:
 - **process** : This method should make 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** : Returns the response with 3rd party site URL to checkout in the **'data'** field of the response. If it 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** /youraccount/upgrade?payment_method=**methodname**&page=**review**&id_package=**premiumpackageid** If you want to show the user what s/he is buying, **page** parameter should be **review** and override **get_transaction_details** , or you may complete the payment after returning from 3rd party site by setting **page** parameter 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).
- The payment methods implemented should be added into **access_control_config.py** .
 - **CFG_CREDIT_CARD_PAYMENT_METHODS** is a dictionary which contains the classes implemented for purchasing with credit card. Its keys are the names 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 names 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** .

<ul style="list-style-type: none"> • Required functions to accommodate billing system have been added into webaccount.py <ul style="list-style-type: none"> – perform_display_payment_complete : Displays the message after buying a premium package. – perform_display_credit_card_form : Displays the credit card form in the buy with credit card page according to active payment gateway. – New argument prem has been added into perform_display_account function to display premium group memberships of the user in Your Account page • Current user group system is changed a little to accommodate premium groups <ul style="list-style-type: none"> – perform_request_leave_group in webgroup.py now checks if the group is a premium group and displays an error message if it is. – get_groups and get_groups_by_user_status functions in webgroup-dblayer.py are checks the membership expiration date of the users. – New functions for premium groups have been added into webgroup-dblayer.py – New group join policy from premium groups ('PG') has been added into CFG_WEBSESSION_GROUP_JOIN_POLICY in websession-config.py • upgrade method has been added into WebInterfaceYourAccountPages in websession_webinterface.py . A user may reach this page when necessary, i.e., visiting a premium collection without corresponding premium group or extending premium group membership. 		
<table border="1"> <tr> <td>Implemented By</td> <td>Alper Çınar (SRDC)</td> </tr> </table>	Implemented By	Alper Çınar (SRDC)
Implemented By	Alper Çınar (SRDC)	

Table 3.33: Implementation Description: RF70

Feature ID	RF71 (Repository Feature 71)
Name	The archive provides a personalized annotating and highlighting tool for users
Effort Spent	4 weeks
Modules Affected / Created	Bibformat, Miscutil, WebSearch, WebSession, WebStyle
Description of the new feature	
<p>An icon has been placed at the right of the page to activate annotating and highlighting feature. This icon can only be visible when a user logs in, otherwise it is not displayed.</p>	



Figure 3.53: RF71 - The icon to activate highlighting

When the icon is clicked, a color palette containing highlight colors appears and previously saved highlighted items are loaded. Four colors are chosen as default. These can be changed from configuration file. 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.

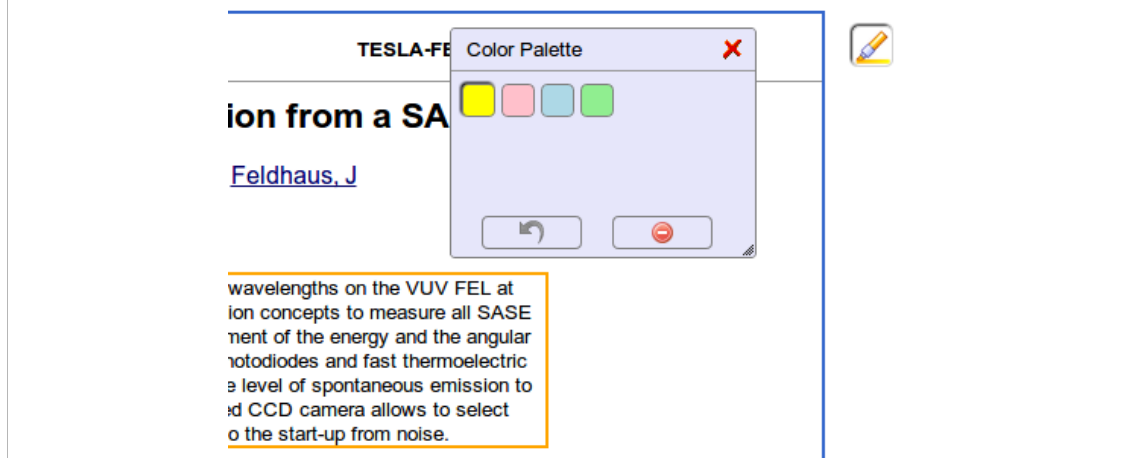


Figure 3.54: RF71 - Color palette

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

principle experiment of self-amplified spontaneous emission (SASE) at short wavelengths on the photon beam diagnostics experiment has been developed employing new detection concepts to measure single pulse basis. The present setup includes instrumentation for the measurement of the energy and the distribution of individual photon pulses. Different types of photon detectors such as PtSi-photodiodes and fast thermographic YBaCuO-films are used to cover some five orders of magnitude of intensity from the level of spontaneous emission at saturation. A 1 m normal incidence monochromator in combination with a fast intensified CCD camera allows to

Figure 3.55: RF71 - Highlighting a part of text

When mouse is rolled over an highlighted area, an **edit icon** is displayed.

photon beam diagnostics experiment has been developed employing new detection concepts to measure a single pulse basis. The present setup includes instrumentation for the measurement of the energy and the distribution of individual photon pulses. Different types of photon detectors such as PtSi-photodiodes and fast thermographic YBaCuO-films are used to cover some five orders of magnitude of intensity from the level of spontaneous emission at saturation. A 1 m normal incidence monochromator in combination with a fast intensified CCD camera allows to

Figure 3.56: RF71 - The icon to edit highlighted text

When **edit icon** is clicked, an **edit menu** providing of 3 options namely **Add/Edit annotation**, **Delete highlight** and **Change color** appears.

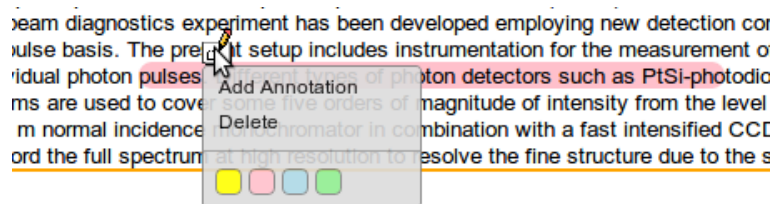


Figure 3.57: RF71 - The edit menu of an highlighted text

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

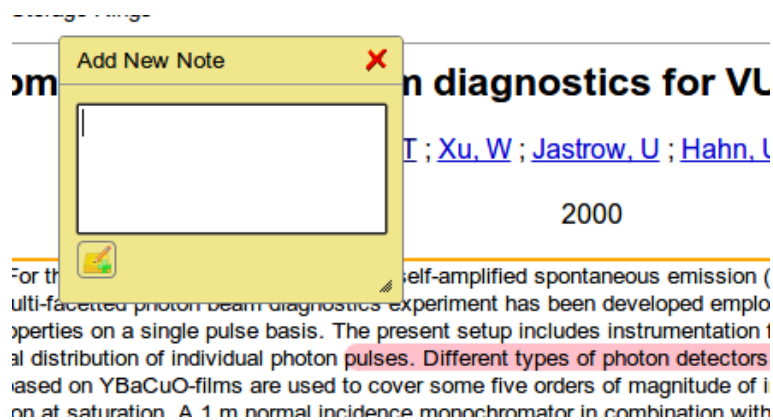


Figure 3.58: RF71 - Adding annotation for a text

After the note is saved, the text of the corresponding highlight becomes shadowed.

diagnostics experiment has been developed employing new detection concept on a single pulse basis. The present setup includes instrumentation for the measurement of the distribution of individual photon pulses. Different types of photon detectors such as PtSi-photodiodes are used to cover some five orders of magnitude of intensity from the level of saturation to the level of single photon pulses. A 1 m normal incidence monochromator in combination with a fast intensified CCD camera is used to select single photon pulses from the background radiation.

Figure 3.59: RF71 - An annotated text

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

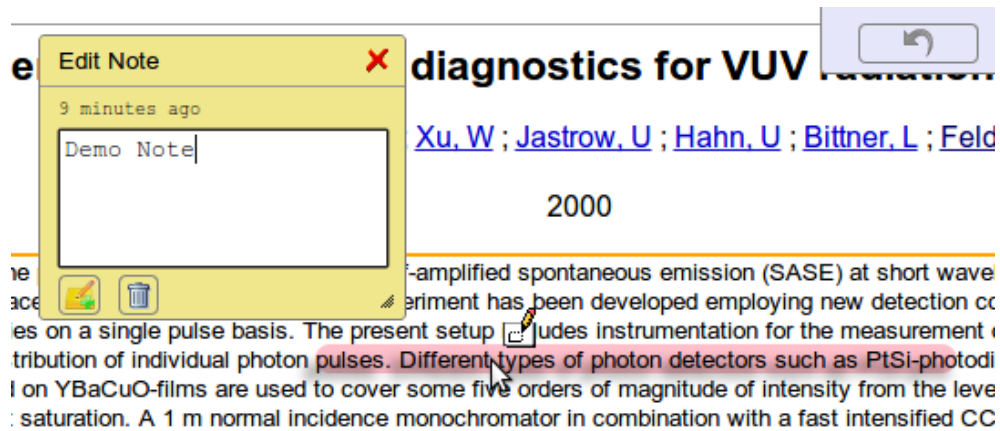


Figure 3.60: RF71 - Displaying/editing an annotation

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

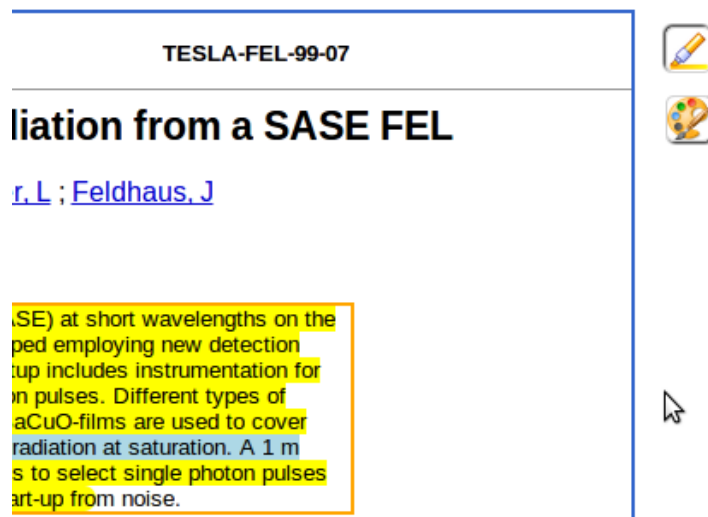


Figure 3.61: RF71 - The icon to retrieve color palette

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 part 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

- 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 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 ids.
- 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

<p>nodes as a result of this selection. This value is also used for annotation id.</p> <ul style="list-style-type: none"> – 'n' denotes the child number that the indices are valid on. – 'c' denotes the color. <p>'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.</p> <ul style="list-style-type: none"> • Each change is directly saved into the database. • 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. • Two new tables namely bibrec_highlights and bibrec_annotations have been inserted into database to keep highlights and annotations. • In websession_webinterface.py, savehighlights, loadhighlights, saveannotation, getannotation and removeannotation methods have been added for communication between server and client sides. • In webuser.py, check_bibrec_modification_date, set_user_bibrec_annotation, get_user_bibrec_annotation, delete_user_bibrec_annotation, set_user_bibrec_highlights, get_user_bibrec_highlights functions have been implemented for database transactions. • In websession_templates.py, tmpl_highlight_tools, tmpl_annotation_box, tmpl_color_palette methods have been implemented to create html codes for highlight tools such as color palette and annotation box. • In dateutils.py, difference_between_times function has been inserted to calculate elapsed time in units such as second, minute, hour etc. • CFG_COLOR_PALETTE parameter has been added to define highlight colors. As default, four colors have been set. • Also in invenioconf.py and search_engine.py, some minor modifications have been made. 		
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%; padding: 5px;">Implemented By</td> <td style="padding: 5px;">Şenan Postacı, Alper Çınar (SRDC)</td> </tr> </table>	Implemented By	Şenan Postacı, Alper Çınar (SRDC)
Implemented By	Şenan Postacı, Alper Çınar (SRDC)	

Table 3.34: Implementation Description: RF71

Feature ID	RF73 (Repository Feature 73)
Name	The archive recommends blogs to users based on the ratings and preferences
Effort Spent	1 Week
Modules Affected / Created	BibRank, WebSearch
<p>Description of the new feature</p> <p>A new ranking method to rank the records by their weighted averages has been developed.</p>	

Sort by: latest first asc. weighted_average Display res: 10 results

Atlantis Institute of Fictive Science

- 1. Ιθάκη / Καβάφης, Κ Π**
(3.108) Σα βγεις στον πηγαιμό για την Ιθάκη,
να εύχεται νάναι μακρύς ο δρόμος,
γεμάτος περιπέτειες, γεμάτος γνώσεις [...]
[Detailed record](#) - [Similar records](#) - [888 reviews](#)
- 2. Supersymmetry and a rationale for sn**
(3.0891) We analyse the CP problem in the context of a s
Fulltext: [PDF](#) [PS.GZ](#);
[Detailed record](#) - [Similar records](#) - [922 reviews](#)
- 3. Filtering Gravity: Modification at Large**
(3.0877) In this lecture I address the issue of possible larg
Published in **Phys. Scr. Top. Issues: T117 (200**
Fulltext: [PDF](#);
[Detailed record](#) - [Similar records](#) - [926 reviews](#)
- 4. Reply to 'Comment on "Solution of th**
(3.0871) This combines a reply to the Comment [hep-th/0
Fulltext: [PDF](#) [PS.GZ](#);
[Detailed record](#) - [Similar records](#) - [920 reviews](#)
- 5. Remarks on the f_0(400-1200) scalar i**
(3.0817) The quark-level linear sigma model is revisited, in
Fulltext: [PDF](#) [PS.GZ](#);
[Detailed record](#) - [Similar records](#) - [920 reviews](#)
- 6. Non-Abelian Flat Directions in a Minin**
(3.0759) Recently, by studying exact flat directions of non
Published in **Mod. Phys. Lett. A: 15 (2000) pp. :**
Fulltext: [PDF](#) [PS.GZ](#);
[Detailed record](#) - [Similar records](#) - [1005 reviews](#)

Figure 3.62: RF73 - Ranking with weighted averages of the records

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

Top Rated Records

- [3.11 Ιθάκη](#)
- [3.09 Supersymmetry and a rationale for small CP violating phases](#)
- [3.09 Filtering Gravity: Modification at Large Distances?](#)
- [3.09 Reply to 'Comment on "Solution of the Relativistic Dirac-Morse Problem"](#)
- [3.08 Remarks on the f_0\(400-1200\) scalar meson as the dynamically generated chiral partner of the pion](#)
- [3.08 Non-Abelian Flat Directions in a Minimal Superstring Standard Model](#)
- [3.08 Neutrino Indirect Detection of Neutralino Dark Matter in the CMSSM](#)
- [3.08 From the standard model to grand unification](#)
- [3.07 Higher Dimensional Schwinger-like Anomalous Effective Action](#)
- [3.07 Nuclear matter with off-shell propagation](#)

Figure 3.63: RF73 - Portalbox displaying top rated records

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

Recently Added Records

- [2012-11-16 This record is just added!](#)
- [2012-11-16 SL\(2,Z\) Action On Three-Dimensional Conformal Field Theories With Abelian Symmetry](#)
- [2012-11-16 Filtering Gravity: Modification at Large Distances?](#)
- [2012-11-16 Closed strings in Misner space: a toy model for a Big Bounce ?](#)
- [2012-11-16 Non-Supersymmetric Membrane Flows from Fake Supergravity and Multi-Trace Deformations](#)
- [2012-11-16 Quasi-normal Modes of Electromagnetic Perturbations of Four-Dimensional Topological Black Holes with Scalar Hair](#)
- [2012-11-16 A new PPN parameter to test Chern-Simons gravity](#)
- [2012-11-16 Matrices on a point as the theory of everything](#)
- [2012-11-16 The wall of the cave](#)
- [2012-11-16 Non-Abelian Flat Directions in a Minimal Superstring Standard Model](#)

Figure 3.64: 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 in which language, 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** .
- **tmpl_top_rated_records_portalbox** and **tmpl_added_content_portalbox** methods have been added into **websearch_templates.py** , which are the templates of the “top rated records” and “added content” portalboxes, respectively.
- 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.

<ul style="list-style-type: none"> • Some modifications have been occurred in <code>bibrank.py</code> and <code>bibrank_tag-based_indexer.py</code> to accommodate new ranking method. • Regression tests for <code>bibrank_weighted_average_indexer</code> and <code>bibrank_archived_content_indexer</code> modules have been added into <code>bibrank_regression_tests.py</code> 		
<table border="1"> <tr> <td>Implemented By</td> <td>Alper Çınar (SRDC)</td> </tr> </table>	Implemented By	Alper Çınar (SRDC)
Implemented By	Alper Çınar (SRDC)	

Table 3.35: Implementation Description: RF73

Feature ID	RF87 (Repository Feature 87)
Name	The archive transforms the SIPS received from the spider to AIP
Effort Spent	1 Month
Modules Affected / Created	BibUpload, BibIngest

Description of the new feature

A new module BibIngest has been developed to host ingestion packages.

The BibIngest module:

- implements the four basic functions of persistent storage: create, read, update and delete (CRUD). Other functions are provided as well, such as validation of the ingestion package etc.
- exposes a single interface that provides the functions mentioned above.
- is able to use a variety of storage engines in the background, as chosen and configured by the administrator.
- is able to transparently support a variety of operations on any of the background storage engines used.
- is extensible enough to accommodate different types of ingestion packages.

Implementation details

In the current implementation of the module the client interface can be found in `bibingest.py`. `bibingest.py` is responsible of instantiating the storage engine (using Invenio's `pluginutils`) and it provides functions on it such as `get_ingestion_package(id)`, `get_many_ingestion_packages(**kwargs)`, `store_ingestion_package(**kwargs)`, `remove_many_ingestion_packages(**kwargs)`, `validate_ingestion_package(content, md5_hash)` etc.

Some of these functions take an arbitrary number of arguments to be translated and parsed by the storage engine. The current implementation expects arguments in the current format: `fieldname.operator = value`. Both `fieldname` and `_operator` are

optional but of course at least one of them should be there. Examples: `id = 5`, `id_in = [1,2,3,4,7,9]`, `id_from = 5`, `id_and_to = 15`, `id_sort_by_asc = True`, `_limit = 20`, etc.

For the time being some configuration options for the module exist in **`bibingest._config.py`**. These include the storage engine name and lists of accepted field names and operators.

An interface class is provided to be inherited and respected by the class implementations of the various storage engines in `filebibingest.engine.interface.py`.

All the storage engine implementations go inside the engines directory (`engines/*.py`). Each engine should define a class that inherits the base class from `bibingest.engine.interface.py` and implements its methods. Since `bibingest.py` is using `pluginutils`, each of these engines should include a global function whose name is the same as the module's name (example: `mongodb.py` should provide the function `mongodb()`). This function should return an instance of the engine's class.

Each engine is responsible for the translation, parsing and implementation of the various fields and operators. An engine has been developed using MongoDB and it is used in the BibUpload post-process described in RF9 to store the SIPs.

Implemented By	Raquel Jiménez, Jaime García, Nikolaos Kasioumis (CERN)
-----------------------	---

Table 3.36: Implementation Description: RF87

Feature ID	RF88 (Repository Feature 88)
Name	The archive stores the content of the AIPS in two different databases for preservation purposes
Effort Spent	1 Week
Modules Affected / Created	BibUpload, BibIngest
Description of the new feature	
<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>	
Implementation details	
<p>Details on how this is done can be found in RF9 and RF87.</p>	
Implemented By	Raquel Jiménez, Jaime García, Nikolaos Kasioumis (CERN)

Table 3.37: Implementation Description: RF88

Chapter 4

Conclusions and Future Work

BlogForever digital repository component is responsible for collecting, preserving and presenting the blog data captured by the spider component. The essential effort to accomplish these tasks was introduced in *D4.4 Digital Repository Component Design* as a list of repository features, and in this report, implementation details of these features planned to be in BlogForever digital repository prototype were presented.

The implementation of the digital repository was planned as enhancements on CERN's Invenio digital library software as iterative cycles rather than implementing a repository from scratch. An agile development method that involves implementation in conjunction with validation was adopted.

To fulfill repository requirements, repository features have been implemented on the vanilla Invenio source. Invenio's modular architecture makes it easily customizable and expandable. In addition, both conveniently modifiable structure of Invenio and easy branching and merging model of Git make the feature based workflow of Invenio applicable to digital repository development. Therefore, same workflow has been adopted in the development of the digital repository, as well.

The features were distributed among the code contributor partners, and each partner is responsible for implementation and documentation of the corresponding features assigned to them. Alongside coding and documenting, the essential tests of each feature is needed to be written as well. Especially, the regression tests have an important role in the integration phase. If necessary, unit tests and web tests were prepared as well to be sure that the feature works without any failures.

The documentation of the features have been carried out through implementation descriptions. An implementation description table has a strict format and introduces the intended result of the feature and describes the technical details of implementation. This document mainly consists of these implementation descriptions prepared by the code contributor partners.

Future work involves implementation of the remaining features, cases studies and validation of the digital repository on new set of weblogs. Moreover, through case studies, WP5 will evaluate the BlogForever platform whether it fulfills the requirements or not and based on the feedback retrieved from platform users, developers will continue to make necessary modifications on the features until a complete system is developed in WP4.

Finally, the outcome of this deliverable D4.5 and outcome of *D4.3: Initial Weblog Spider Prototype*[5] will be integrated and finalized in *Task 4.6 Integration and Standardization* yielding the final BlogForever platform.

References

- [I] BlogForever. Description of work, 2011.
- [II] Git Workflow in Invenio. <http://invenio-software.org/wiki/tools/git/workflow>, [Retrieved January 15, 2012].
- [III] H. Kalb, N. Kasioumis, J. García Llopis, Şenan Postacı, and S. Arango-Docio. D4.1: User Requirements and Platform Specifications. Work package, Technische University at Berlin (TUB), December 2011. Work Package Four Deliverables.
- [IV] J. García Llopis, R. Jiménez Encinar, K. Stepanyan, Y. Kim, A. Haberfeld, Şenan Postacı, G. Gkotsis, P. Lazaridou, A. Çmar, H. Kalb, V. Banos, S. Arango Docio, N. Kasioumis, T. Šimko, G. Banu Laleci, and E. Pinsent. D4.4: Digital Repository Component Design. Work package, European Organization for Nuclear Research (CERN), November 2012. Work Package Four Deliverables.
- [V] M. Rynning. D4.3: Initial Weblog Spider Prototype. Work package, CyberWatcher, September 2012. Work Package Four Deliverables.
- [VI] P. Sleeman, S.Arango-Docio, E. Pinsent, G. Gkotsis, T. Farrell, S. Kopidaki, and M. Rynning. D5.1: Design and Specification of Case Studies. Work package, University of Londond (UL), June 2012. Work Package Five Deliverables.

