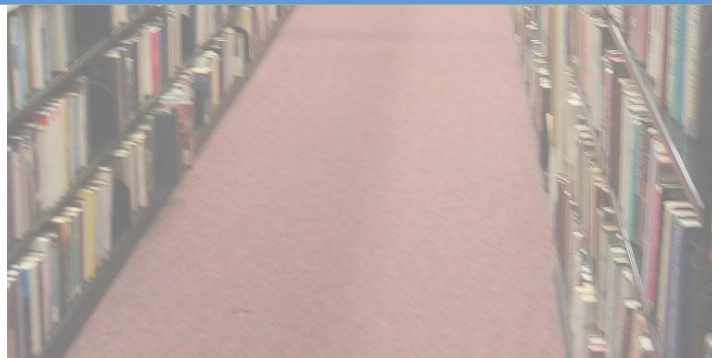


April 21, 2009



# INTERACTIVE EDITING AND CATALOGING INTERFACES FOR MODERN DIGITAL LIBRARY SYSTEMS



## **Bachelor Project Report**

Bergen University College  
Faculty of Engineering  
Department of Computing

Lars Christian Raae

## Preface

This is the mandatory final report of my bachelor project which I have completed at CERN, where I have been staying as a Technical Student in the period from May 2008 until the end of June 2009. The topic of my project is *Interactive Editing and Cataloging Interfaces for Modern Digital Library Systems*. This is a broad subject which is thoroughly discussed in chapter 1. My contribution is a major upgrade of a system module called *BibEdit*, to prepare it for a central role in the cataloging workflow of a new High-Energy Physics information system called *INSPIRE*.

This is not the only assignment I will have had in my 14 months at CERN, but it is the one I have chosen to focus on as my bachelor project. The development commenced in September 2008 and as I'm writing this report in April 2009, it will focus on the progress made in the last 8 months, although I hope to keep developing and improving BibEdit until the end of my stay here at CERN.

This report serves as the primary written documentation of my bachelor project and encompasses the content of my pre-project report which I wrote and presented in the early phase of the project. The project itself is the completion of my bachelor's degree in Computer Engineering at Bergen University College, which I started in the autumn of 2005. Initially a three year degree, I'm now at the end of my fourth year because of my choice to participate in CERN's Technical Student Programme together with another student from my class. In comparison, our classmates chose projects of ordinary length (approximately two months) and received their degrees already last summer.

So why postpone the completion of our degree with a full year voluntarily? Why leave everything safe and familiar behind to go work in a foreign country with a foreign language? Why move to a strange place like CERN, mostly famous for housing a huge subterranean tunnel where some, possibly dangerous, (if one should believe the tabloids) experiments are being performed?

I will introduce CERN properly in the report, but suffice to say it is probably the most advanced research laboratory in the world. And although so big that it's located partly in Switzerland and partly in France, it's not a place where everyone gets to go work but, because of several lucky breaks, we did. Personally I had even more reasons to go. One of them was an ambition to learn a new language - French. Moreover I always wanted to travel abroad and do part of my education in a different place with a different culture. Needless to say when we got this chance, we hardly needed time to think about it.

The objectives of this report are as follows;

- to present the background of my project; CERN and my employers here, the INSPIRE project, the work of a cataloger and the BibEdit tool
- to walk through the different phases of the system development process
- to discuss the management of the project; people, time and tasks
- to give a brief conclusion

The report is written over a period of approximately three weeks, to be finished in time for my project presentation over the Easter holidays. I hope that is just enough time to make it into a quality product. It is, however, a rather large report as it tries to summarize about 8 months of full-time work on the project in a systematic way.

The main audience for this document is my academic supervisors at Bergen University College and my employers and colleagues at CERN, the latter are one of the reasons that I've chosen to write in English. Secondary comes anyone with a technical or personal interest in my work, but the report is written in a fairly technical language and as such requires a background in computing to be fully appreciated. Lesser known vocabulary, technical or otherwise, is explained in the dictionary towards the end of the report.

I continue to enjoy my work at CERN, my dear friends and colleagues, and the part of France and Switzerland where I currently live, very much! If it hadn't been for personal obligations in Norway I might have considered applying for another position here.

The work at CERN has given me practical real-life experience as a programmer and as an employee in a large, technically advanced organization. I'm sure I will profit from having worked with modern and increasingly popular technology such as Python and Ajax. I've also had the honor of working on the INSPIRE project, which might spell the beginning of a revolution in the way we distribute and disseminate scientific information in the field of High-Energy Physics and maybe eventually in other fields as well.

My project blog is available at: <http://raae-bachelor.blogspot.com/>

I would like to extend my sincerest thanks to the people I've been depending on both professionally and personally for the last year:

- My contact at Bergen University College: Professor Håvard Helstrup
- My supervisor in the CERN Scientific Information group: Jens Vigen
- My supervisor in the CERN IT CDS section: Jean Yves Le-Meur
- Lead developer and tech guru for CDS Invenio and INSPIRE: Tibor Simko
- Travis C. Brooks of Stanford Linear Accelerator Center, SPIRES and the INSPIRE collaboration
- All my current and previous colleagues and friends in the CDS section

And last, but not least, my family and my girlfriend for being there for me, even when I was far away.

## Table of Contents

1. Introduction.....	4
1.1 Employer.....	4
1.1.1 CERN .....	4
1.1.2 My employer: The CERN Scientific Information group .....	5
1.1.3 My place of work: The CDS section .....	5
1.1.4 My project group: INSPIRE .....	5
1.2 Project background .....	6
1.2.1 CDS Invenio.....	6
1.2.2 SPIRES .....	7
1.2.3 INSPIRE .....	7
1.2.4 Open Access .....	8
1.3 The project .....	9
1.3.1 BibEdit - a record editor .....	9
1.3.2 Cataloging and MARC .....	9
1.3.3 Cataloging in SPIRES and INSPIRE.....	10
1.3.4 BibEdit in INSPIRE .....	11
1.4 Project methodology .....	12
2. Requirements specification.....	13
2.1 Changes to the original requirements.....	13
2.2 Use cases .....	14
2.2.1 Informal use case.....	14
2.2.2 BibCheck - Dynamic Functionality Use Case #1.....	14
2.2.3 BibCheck - Dynamic Functionality Use Case #2.....	14
2.3 Functional requirements .....	15
2.3.1 Functional requirements .....	15
2.3.2 Non-functional requirements.....	16
3. Analysis.....	17
3.1 The INSPIRE cataloging tools .....	17
3.2 Solution alternatives.....	18
3.2.1 The classic web tool.....	18
3.2.2 The Ajax web tool .....	19
3.2.3 The desktop application .....	20

3.2.4 Discussion of the options .....	20
3.3 Evaluation and selection of tools and programming languages .....	22
3.3.1 Technologies – overview .....	22
3.3.2 CDS Invenio and the LAMP software bundle.....	23
3.3.3 Source code management system: Git.....	23
3.3.4 JavaScript framework: jQuery .....	24
3.3.5 Data transfer protocol: JSON.....	25
3.3.6 Code editor: Emacs.....	26
4. Design .....	27
4.1 Architecture.....	27
4.2 Design solutions .....	29
4.2.1 Browser compatibility .....	29
4.2.2 URL and navigation.....	29
4.2.3 Diff and merge.....	30
4.3 Graphical User Interface.....	31
4.3.1 Mock-up screenshots – early design stage .....	31
4.3.2 Mock-up screenshots – advanced design stage.....	32
4.3.3 GUI layout.....	35
4.3.4 In-place editing.....	36
4.3.5 Optimizing GUI performance .....	36
4.3.6 Streamlining the GUI .....	37
5. Implementation .....	39
5.1 Ajax .....	39
5.1.1 Ajax requests with jQuery and JSON .....	39
5.1.2 The URL state manager .....	40
5.2 GUI.....	42
5.2.1 HTML, CSS.....	42
5.2.2 IDs and classes.....	42
5.2.3 The left-side menu.....	44
5.2.4 In-place editing.....	47
5.2.5 In-place adding of fields .....	48
5.2.6 Hotkeys.....	49
6. Testing .....	51
6. 1 The users.....	51

6.2 User testing .....	51
6.2.1 Design stage .....	51
6.2.2 Development stage .....	51
6.3 Handling of feedback and bugs .....	52
6.4 Automated tests .....	52
6.5 Release routine.....	53
7. Project management .....	54
7.1 The project group .....	54
7.2 Time management .....	54
7.3 Task management .....	55
8. Conclusion .....	56
9. Literature .....	58
9.1 References .....	58
9.2 Web sites of interest .....	61
9.2.1 Project and background material .....	61
9.2.2 Technical resources .....	62
9.3 Books .....	62
Appendix A: Project schedule.....	63
A.1 Deliverables.....	63
A.2 Gantt .....	63
Appendix B: Risk Assessment .....	64
Appendix C: User Documentation .....	65
BibEdit Keyboard Shortcuts.....	65
Basic record actions.....	65
Focused (clicked) subfield .....	66
Input field/form.....	67
Other functionality .....	67

# 1. Introduction

## 1.1 Employer

### 1.1.1 CERN

The field of experimental particle physics probes the fundamental structure of matter. It is also called high energy physics, because many elementary particles can only be created during energetic collisions of other particles. Today we search for these particles using *particle accelerators* to collide particles together at very high energy levels and *detectors* to detect particles in the debris of the collision [1].

CERN is located on the Franco-Swiss border near Geneva. It is the world's largest particle physics laboratory, established in 1954 as a joint venture between several European countries. It now has 20 Member States, among them Norway [2]. CERN has approximately 2 600 fulltime employees, as well as some 8 000 visiting scientists and engineers, representing more than 500 universities and 80 nationalities [3].



Figure 1: The CERN Meyrin main site

The 10<sup>th</sup> of September 2008, in an event followed closely all over the world, the first particle beam circulated in the Large Hadron Collider (LHC). The LHC is a near 27 km long tunnel of superconducting magnets, built to collide particles at extremely high levels of energy. When it is operational it is expected to confirm (or perhaps invalidate) existing theories and produce breakthrough discoveries in the field of particle physics. Unfortunately the machine was forced to shut down after a serious accident only nine days after the startup. It is now being repaired and is due to restart in the autumn of 2009 [4].

### **1.1.2 My employer: The CERN Scientific Information group**

My employer at CERN is the Scientific Information (SI) group, which has the CERN library as one of its responsibilities. There is also a section in this group dedicated to the work on open access (see chapter 1.2.4). The SI group has hired many people, including other technical students like myself, to work on development related to digital library systems.

Though the SI group isn't actually where I have performed my daily work (see chapter 1.1.3), I keep close ties with them. My supervisor is in this group, I attend their group meetings (though they are in French - the group funds my French courses in an attempt to reduce the language gap) and as any member of this group I spend at least two hours each week working at the library desk. Additionally many of the users of my application belong to this group, so I often visit them to discuss design decisions, present progress or talk about issues that have come up.

### **1.1.3 My place of work: The CDS section**

Though my employer is SI, for most practical purposes I function as part of the IT department's CDS section. I share office with members of the CDS section and they are the people I work most closely with. I report my progress in weekly section meetings and discuss technical matters with our lead developer or other colleagues.

CDS is short for CERN Document Server and is also the name of a large CERN information system that this section is responsible for maintaining technically. The CERN Document Server manages around 900 000 bibliographic records and 350 000 full text documents, covering preprints, articles, books, journals, photographs, videos and more, of interest to people working in particle physics [6]. More importantly in this context; the section is responsible for the development of the CDS Invenio software which runs the CERN Document Server (see chapter 1.2.1).

### **1.1.4 My project group: INSPIRE**

The INSPIRE collaboration is the result of an agreement between four major particle physics laboratories. The INSPIRE project group consists mainly of people working with scientific information, open access or software development and have representatives from all the participating labs. We have semi-weekly phone meetings or videoconferences and have also had two weeklong workshops during my project period. On a daily basis the group communicates using e-mail and the collaboration wiki tool *Twiki*.



## 1.2 Project background

### 1.2.1 CDS Invenio

The CDS section develops a software package called *CDS Invenio* (sometimes just referred to as *Invenio*); «a suite of applications which provides the framework and tools for building and managing an autonomous digital library server» [5]. CDS Invenio is GPL licensed, uses MARC 21 as its underlying bibliographic standard and is deployed at more than 20 institutions world-wide. The CERN Document Server runs on the CDS Invenio software.

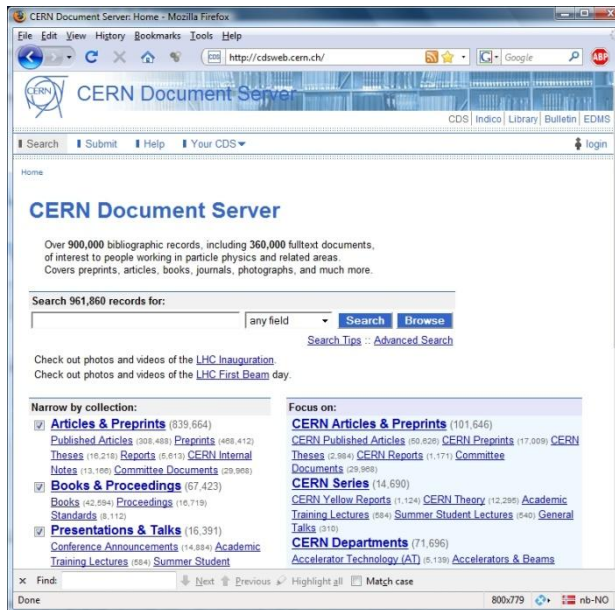


Figure 2: The front page of the CERN Document Server

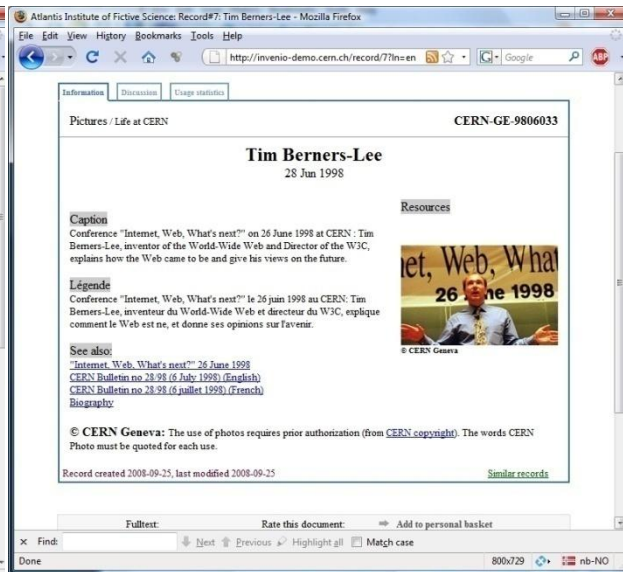


Figure 3: A photo record in CDS (HTML detailed format)

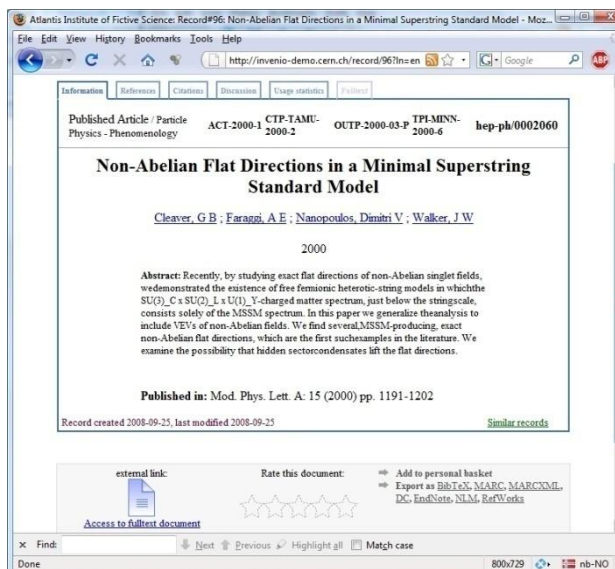


Figure 4: An article record in CDS (HTML detailed format)

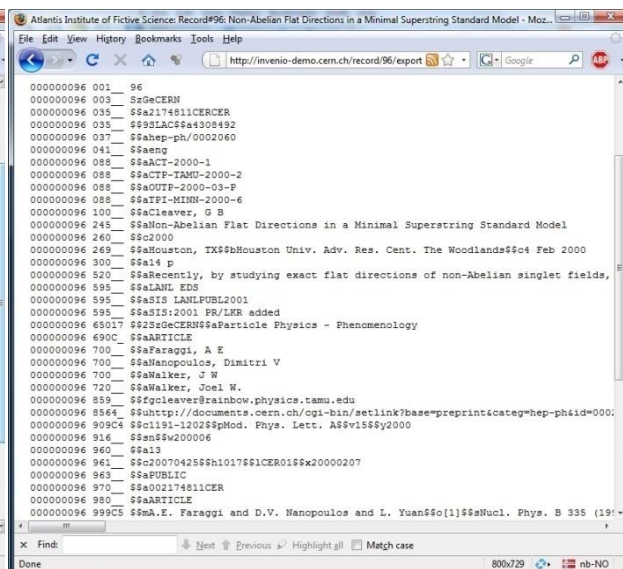


Figure 5: An article record in CDS (MARC format)

### 1.2.2 SPIRES

The SPIRES High Energy Physics database (today usually just referred to as SPIRES) stores bibliographic information about the literature of the field of High Energy Physics. It is hosted at SLAC, the Stanford Linear Accelerator Center in California, and jointly compiled together with DESY, the Deutsches Elektronen-Synchrotron in Hamburg, Germany, and Fermilab, the Fermi National Accelerator Laboratory in Illinois [7].

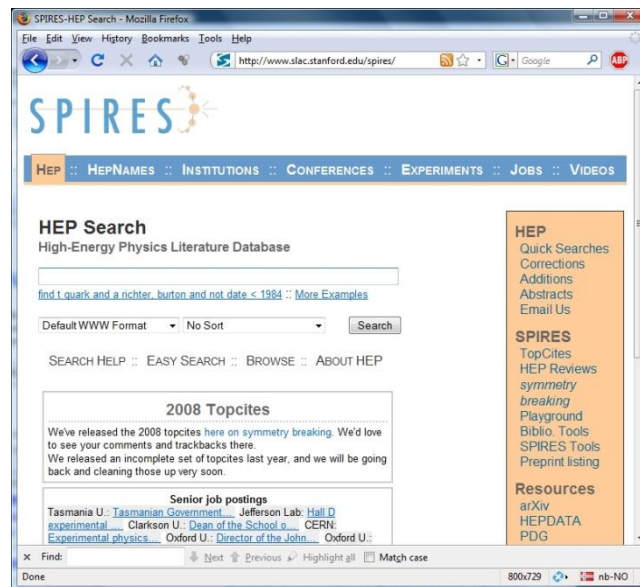


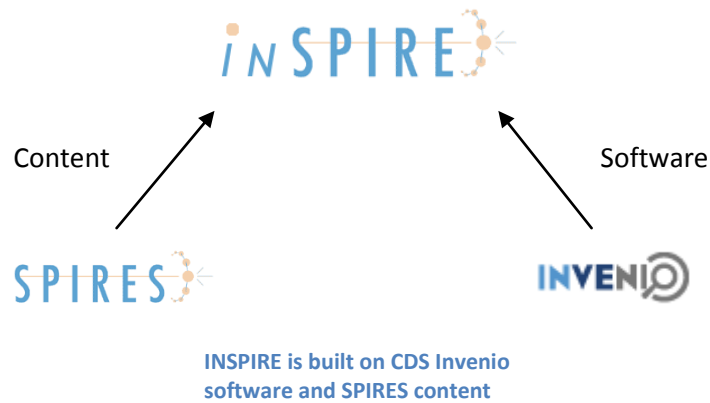
Figure 6: The front page of SPIRES

SPIRES is an indispensable tool for most HEP physicists and today functions as a gateway to all information in this field of research. The database engine running SPIRES has its roots back in the 1960s and SPIRES had the honor of becoming the first database available on the World Wide Web in 1991 [8]. Nevertheless being such a veteran system also has its price, like complicating maintenance and further development, so in the recent years one has been searching for a new platform to host the content of SPIRES [9].

### 1.2.3 INSPIRE

In May 2007 CERN, DESY, Fermilab and SLAC announced that they would join forces to build INSPIRE, «the next generation High Energy Physics (HEP) information system». INSPIRE is being built by combining the SPIRES database with the CDS Invenio software. However INSPIRE will not only be the SPIRES data moved to a new and better platform. The press release states that INSPIRE «will develop long-sought features, providing access to the entire corpus of the HEP literature with full-text Google-like search capabilities and enabling innovative text- and data-mining applications» [10].

A lot of the new functionality that INSPIRE will provide is introduced simply by using the already very feature rich CDS Invenio software as platform, for example web 2.0 style personalization and collaboration tools like user defined baskets and e-mail alerts, and commenting and reviewing. CDS Invenio also features a powerful search engine, customizable collections trees and the tools needed to easily harvest data from other systems using recognized and open protocols such as OAI [11].



#### 1.2.4 Open Access

The vision is that INSPIRE will be *the* global information system for the HEP community world-wide featuring not just bibliographic data, but also providing open access to full text articles and other scientific information. The idea of open access is a major trend in scientific publishing today and is one of the things that makes INSPIRE such an interesting project to work on.

Traditionally scientific journals have charged subscription fees to fund the administration of the peer reviewing process, the cost of publishing and often to make a profit. A consequence of this is that scientific journals usually are not freely available and even libraries and research institutions are often forced to choose a limited set of journals because of the high cost of subscriptions. A perhaps more principal problem is that the results of publicly funded research are not publicly available.

Open access publishing is instead, as Wikipedia writes: «...also known as "author-pays" or "paid on behalf of the author", where a publication charge is paid by the author, his university, or the agency which provides his research grant» [12]. In the field of HEP the problem is attempted solved by establishing a Sponsoring Consortium for Open Access Publishing in Particle Physics (SCOAP<sup>3</sup>), where funding bodies and libraries together pay for the peer-review service to make articles freely available for anyone to read [13].

## 1.3 The project

### 1.3.1 BibEdit - a record editor

The primary task of digital library systems such as CDS Invenio is to store bibliographic information, metadata, about information entities. The entities themselves can be just about anything. Books are maybe the first thing that comes to mind, because of the *library* term, but it can just as well be documents, presentations, multimedia or whatever you would like to store information about. As digital storage becomes increasingly common, so does the situation when the entity itself is also stored in the system, for example the full text of an article, a photo or an e-book.

The bibliographic information of an object is called its *bibliographic record* and a basic tool for CDS Invenio and similar systems is, of course, a *record editor* to let you enter or make corrections to a bibliographic record. The record editor in CDS Invenio is called *BibEdit*.

The BibEdit module existed before I started my project. It was developed as a summer project (or over a comparably short period of time) by a temporary developer some time ago. The existing version featured the basic minimal functionality one would expect, which was to let you edit the data of an existing record.

### 1.3.2 Cataloging and MARC

A cataloger (or cataloguer) can be defined as «a library professional who assigns call numbers, subject headings, and otherwise constructs the entire bibliographic record of an item» [14]. Sometimes the word *curator* is used instead of *cataloger*, but in our context the meaning should be mostly the same: a person responsible for the verification and maintenance of a collection of bibliographic information.

CDS Invenio follows the *MARC 21* format for bibliographic data, developed by the U.S. Library of Congress. Because of this the data is often loosely referred to as *MARC* or *the MARC*. Here is my own very quick and unofficial introduction to MARC, starting with part of an example record from CDS Invenio's demo records, a book called *Electrical breakdown in gases*:

```
001__ 34
245__ $$aElectrical breakdown in gases
270__ $$ged.$$pRees, J A
980__ $$aBOOK
```

Each line starts with a three character sequence called the *tag*, for example *001* or *980*.

Each line describes a *field*. The first field is a *controlfield*. A controlfield has a tag between 001 and 009 and then some content; in this case the content is *34* which is the record identifier.

The rest of the fields are ordinary data fields. After their tag follow two characters called *indicators*, though in this case none of the fields have them specified and the underscores simply denote that they are empty.

The two dollar symbols denote the start of a *subfield*. The first character after the dollar symbols is the name of the subfield. After this follow the content of the subfield. For example we see that field 245\_\_ has a subfield *a* with content *Electrical breakdown in gases*, which is the title of the book. We also see that the field 270\_\_ has two subfields – *g* and *p*.

Although this example might look simple and strict, MARC is a very relaxed format in the sense that it gives you a lot of freedom in deciding how you want to catalog your collection, for example:

- A tag may contain several fields and a field may contain several subfields.
- Fields may have duplicates. Subfields may have duplicates within their field.
- The order of fields matter. The order of subfields within a field matters. (Such is our implementation anyway, the MARC format might not be entirely clear on this).

Though this freedom to customize the format to suit your needs might be appreciated by the average cataloger, it might also lead to some bad archiving practices if used insensibly. And what's certain is that it demands a lot from computer tools that does any form of MARC manipulation.

### 1.3.3 Cataloging in SPIRES and INSPIRE

In SPIRES, and INSPIRE when it is finished, most records come to existence either through manual inputting, meaning that someone enters the data (this might be a cataloger, the author of the document or some other “external” party), or through harvesting, meaning that some process fetches a batch of records from an outside source or system. No matter what the source of the record, it is not allowed to go into the system without some human intervention.

The catalogers at the laboratories participating in the INSPIRE project will have the important job of controlling, cleaning and enriching the flood of data going into the finished system. So far they have been working on separate systems and therefore doing a lot of duplicate work, something that INSPIRE should put an end to.

Before any record can go into the system a cataloger must review it, until that happens it is placed in a holding pen from where a cataloger eventually fetches it. The cataloger asserts that the record is actually relevant to the system. He/she then wants to determine if it's a duplicate of a document that already exists in the system. If it is, it still might contain information not present in the existing copy, in which case the two articles should be merged into one.

After this the record data is checked for correctness and possibly altered so it's in accordance with agreed upon standards for archiving in the system. Some parts of it go through special validity checks, such as authors and institutions which are matched against existing databases (also called knowledge bases or authority files). Other tools have automatically extracted keywords and references in advance. The cataloger must verify that these are correct.

### 1.3.4 BibEdit in INSPIRE

In INSPIRE, to streamline the workflow described in chapter 1.3.3, a range of modules and tools will play their parts to meet the following objectives;

- enable global cooperation between catalogers at the different labs
- automate as much of the catalogers work as possible
- provide tools to let the catalogers do their job easily, efficiently and without needlessly repetitive or pointless tasks

An interface is needed that gathers the results from all these modules and displays them to the cataloger in the context of the record, allowing him/her to make the final decisions on how the record will look. It would be ideal to tie this interface together with the basic record editor, BibEdit, so that the display and edit mechanisms already present here can be reused and enrich the functionality of the interface even more. This describes in brief the objective of my bachelor project.

The goal of the “upgraded” BibEdit tool is to provide a GUI for interactive record editing, enrichment, correction and verification functionality. Using the editor, the user should be quickly able to correct typos, assign correct (canonical) institute names/codes in the record from authority records, use the canonical form of the author's name, the publication's name, the item's language etc.

The philosophy is to make the common cases as efficient as possible, and to have the data do as much work as possible before the cataloguer sees/touches it. The automated tools should be able to check most things and make guesses in common cases that the cataloguer can confirm. The number of actions and keystrokes by the inputter should be kept to an absolute minimum.

The developments should be done as direct enhancements to the CDS Invenio software, since most of them will be applicable to other systems than INSPIRE. In the situation that some feature is relevant only to INSPIRE, there are ways to avoid its inclusion in other systems.

## 1.4 Project methodology

The project development is being done as improvements to the CDS Invenio software. The project therefore follows the CDS Invenio development practices as described on the CDS Invenio Twiki at <https://twiki.cern.ch/twiki/bin/view/CDS/Invenio>. To quickly summarize:

- Rather informal requirements and design phases; talks with users, paper prototyping, sketching and discussions using email and Twiki.
- “Code as design” approach.
- Imperative/functional programming or simple OOP preferred for performance reasons.

To minimize most of the potential risks (see appendix B) the basic functionality will be designed and implemented first, in essence replacing the existing version of BibEdit. During this part it will become clear if the chosen solution is feasible and can at all provide acceptable usability.

Once the basic tool is ready we can start discussing and prioritizing the new features that need to be finished before the application can be put in production. From this point on development will be done on a feature by feature basis with a few features being designed and implemented over short periods, with a continuous evaluation of what should be given priority and included in the next feature package. This is near to the SCRUM style of iterative development.



## 2. Requirements specification

### 2.1 Changes to the original requirements

There have been major changes to the original project requirements since I started working. Large parts of the original BibEdit specification have been made into separate components or subcomponents and assigned other developers.

Initially BibEdit was to provide a merging interface that would show the differences between two (or maybe more) records in an understandable and graphic way, and let the cataloger easily copy changes between them, to quickly merge them into one. It soon became clear that this would demand too much flexibility from the BibEdit tool, which already had a large amount of features in the requirements, and that everyone would profit if the merging functionality was made into an individual tool. This module was called BibMerge and the project was assigned another developer.

BibEdit was also supposed to have a “multi edit” mode, where you could perform changes to a set of records at the same time, for example to correct a typo present in a hundred or more records or to add a new field to all records of a certain type. It soon became obvious that the interface needed to specify such changes would probably not look very similar to the single record editor and therefore *BibEdit MultiEdit* was also made into a separate tool, sort of a subcomponent of BibEdit.

And finally, at one point in the project we saw that while BibEdit could function excellent as a general record editor, a lot of the catalogers work would be separated into the verification and cleaning of three types of data: authors, keywords and references. Each data type should be presented in a unique way and allow for a few fast and simple operations to maximize the efficiency of the cataloger. To avoid cluttering up the general editing interface with this very specific functionality it was decided to start developing *BibEdit special modes* to edit these three types of data. These modes were also separated from my project and reassigned another developer.

It now seems clear, from the work that has since been done by me and the other developers who have become involved, that my original project specification might have been a bit overwhelming, even for an *extended* bachelor project.

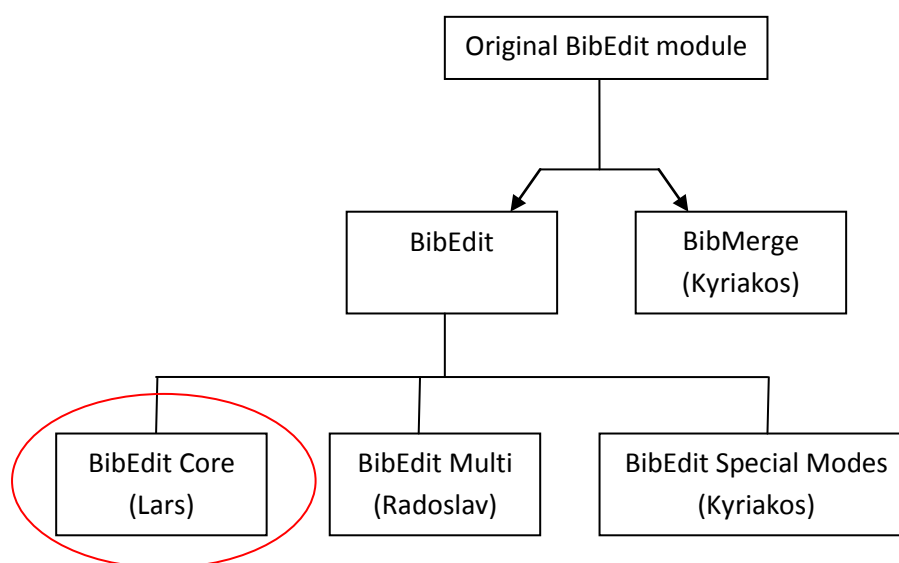


Figure 7: The dividing of the original project



## 2.2 Use cases

As provided by the INSPIRE collaboration.

### 2.2.1 Informal use case

A set of new records have been entered or harvested. They are in a holding pen. Checking the set has been assigned to the cataloguer. He/she logs in and sees a list of the set's records.

The cataloguer checks the records. Before invoking BibEdit the BibMerge module, provides a GUI for eliminating or merging duplicate records. Some records are duplicates or represent items of no interest -- they are deleted by the cataloguer. Then BibEdit is called together with BibCheck on each record. For records that do not need improving he/she uses the 'save'-function and the record is saved in the production database.

For record X, the institute name is misspelled / unknown. The cataloguer selects the institute field and selects the correct name from a pop-up list.

The language field for the record is missing. The cataloguer selects the correct language in a pop-up list.

One of the references of the record has a misspelled journal name. The cataloguer selects the correct name in a pop-up.

Keywords are missing. The cataloguer selects them in a pop-up list.

### 2.2.2 BibCheck - Dynamic Functionality Use Case #1

**Typing something in the field that you want to use as a lookup for the true value to be inserted**

I want to insert institution code 14567 (which is "SLAC, SSRL")

I don't know the code (neither the name nor the number).

I type (in inst. field) "Menlo Park CA" and BibKnowledge returns a list of insts. that have this address ("SLAC", "SLAC, SSRL", "KIPAC, Menlo Park"). I pick from this list and the resulting code name is stored in the field.

This is doing a (sophisticated) lookup to determine the correct value of 1 field, based on user input.

### 2.2.3 BibCheck - Dynamic Functionality Use Case #2

The second case is maintaining something like a field that depends on another. In the above case, we might imagine storing the inst. numerical code and the inst. name (100\_i 100\_u respectively). If the numerical code is present, then we should be able to edit that field and see the 100\_u change based on the 110a value in the inst record having a code = 100\_i. Possibly one might like this to work in the opposite direction as well? Or perhaps it is saner to define a "primary" field and a set of secondary/derived fields.

## 2.3 Functional requirements

### 2.3.1 Functional requirements

Category	Description	Priority
Core	Fetch and display record by record ID	High
Core	Submit a changed record to the database	High
Core	Cancel editing	High
Core	Delete the record	High
Core	Fetch and browse a set of records by custom search query	Medium
Core	Create a new empty record	Medium
Core	Create an empty record from a template	Medium
Core	Clone a new record from an existing record	Medium
Editing	Add a new controlfield	High
Editing	Add a new field with one or more subfields	High
Editing	Add a new subfield to an existing field	High
Editing	Add a new controlfield or field from a list of field templates	Medium
Editing	Edit the content of a controlfield	High
Editing	Edit the content of a subfield	High
Editing	Edit the MARC of a controlfield, field or subfield	Medium
Editing	MARC validation of input	Medium
Editing	Move a controlfield up or down in its field group	High
Editing	Move a field up or down in its field group	High
Editing	Move a subfield up or down in its field	High
Editing	Delete a controlfield	High
Editing	Delete a field	High
Editing	Delete a subfield	High
Editing	Select multiple controlfields and/or fields and/or subfields	High
Editing	Delete multiple controlfields and/or fields and/or subfields	High
Editing	Undo last / last few edit actions	Medium
Editing	Copy, cut & paste fields or subfields Care must be taken to illegal paste operations.	Medium
Editing	Protected fields or field groups Fields or field groups specified as protected in the Invenio configuration should not be allowed added, modified or deleted through BibEdit.	Medium
View	Toggle between display of tags as MARC or human labels	High
BibCheck	Check a full record	High
BibCheck	Automatic application of corrections	High
BibCheck	Display of warnings and errors	High
BibKnowledge	Autocomplete for fields with lookups in auth. records	High
BibKnowledge	Verification of fields with lookups in auth. records	High
BibKnowledge	Option to register RT task for entries missing in auth. records	High
History	Fetch an earlier revision of a record	High
History	Compare two revisions of a record (possible utilizing BibMerge)	High
History	Revert to an earlier revision of a record	High

### 2.3.2 Non-functional requirements

Category	Description	Priority
Availability	BibEdit must be available to catalogers at all the four labs participating in the INSPIRE project, regardless of operating system or other reasonable platform restrictions.	High
Interface	BibEdit must be callable with simple custom arguments For example must BibCatalogue be able to launch BibEdit on a specific record.	High
UI	Existing command line functionality should be conserved	High
UI	Hotkeys It should be possible, maybe even preferable, to operate the application purely by using the keyboard.	High
UI	Internationalization CDS Invenio has multilingual support, so should the BibEdit module.	Medium
UI	All basic editing functionality for a record should be accessible from one screen	High
Performance	Common edit operations like adding, editing or moving fields around should have average processing times of less than one second.	High
Performance	Less common operations like loading or submitting a record should have average processing times of less than three seconds.	High
Security	The changes a user does before submitting the changed record should be stored at a temporary location, so that no significant work will be lost if the application crashes.	High
Security	Only users with explicit authorization (as general or collection curators) should be allowed to open a record in BibEdit	High
Security	Locking of records that are being edited or that are in the upload queue Until a mechanism is in place to detect, submit and merge differentials (backend code for this is not part of BibEdit) it is unsafe to let two users edit a record simultaneously, so the first user to open the record should also lock it. Neither is it safe to edit records that have new versions pending in the upload queue. These should also be locked.	High
Security	Logging of user edits It should be possible to say from the logs which user submitted a given revision of a record.	Medium
Help	Provide users with a shortcut to the BibEdit admin guide	High
Help	Expand admin guide with basic usage instructions In particular pertaining to use of mouse and keyboard.	High

## 3. Analysis

### 3.1 The INSPIRE cataloging tools

I will do a quick rundown of the components involved in the INSPIRE cataloging workflow (see chapter 1.3.3 and 1.3.4) to put my project in the appropriate context.

**BibCatalogue** is the starting point for any cataloging task. It is being implemented with a request tracking system called RT. The idea is that every task or request is registered in RT and assigned to a cataloger with the correct responsibility, competence and authorization. This can be a fully automatic process or some human intervention could be required. From the RT task the cataloger can launch the different tools he/she needs to carry out the task.

**RefExtract** and **BibClassify** are tools that serve to automatically extract information from articles or other documents. RefExtract extracts and parses lists of references, while BibClassify tries to intelligently determine suitable keywords to describe the document through various clever analysis and algorithms.

The **BibCheck** module performs automatic checks on records based on a predefined set of rules. It returns with a list of errors or warnings (requiring human intervention) and fixes (which can either be applied automatically or reviewed by a human first).

The **BibKnowledge** module provides Web tools for catalogers to manage various knowledge bases, authority files, and taxonomies or thesauri. These databases contain information needed for standardization and record quality checking.

Except for BibCatalogue all the modules described here are support modules that provide some kind of data to enrich or refine a record. BibEdit is the tool that must integrate and present the data from all these modules; lookups and suggestions from BibKnowledge, corrections and errors from BibCheck and extracted keywords and references from BibClassify and RefExtract.

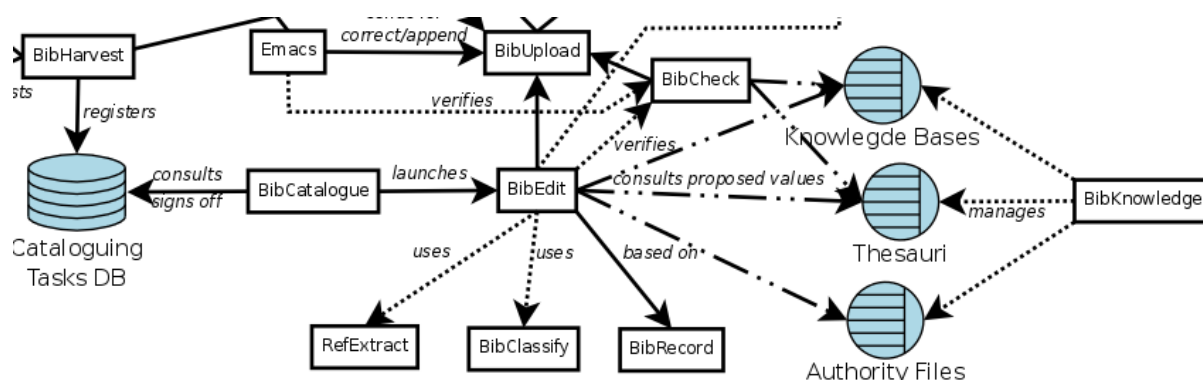


Figure 8: The role of the BibEdit module in INSPIRE

## 3.2 Solution alternatives

### 3.2.1 The classic web tool

BibEdit already exists as an Invenio module. This module is constructed like most Invenio modules, characteristics being;

- HTML pages generated by several layers of serverside Python scripts.
- User interface consisting of static HTML pages.
- User interaction consists of clicking links with customized URL queries or submitting HTML forms. These actions invoke some kind of serverside processing, which eventually results in a new page being delivered to the user.

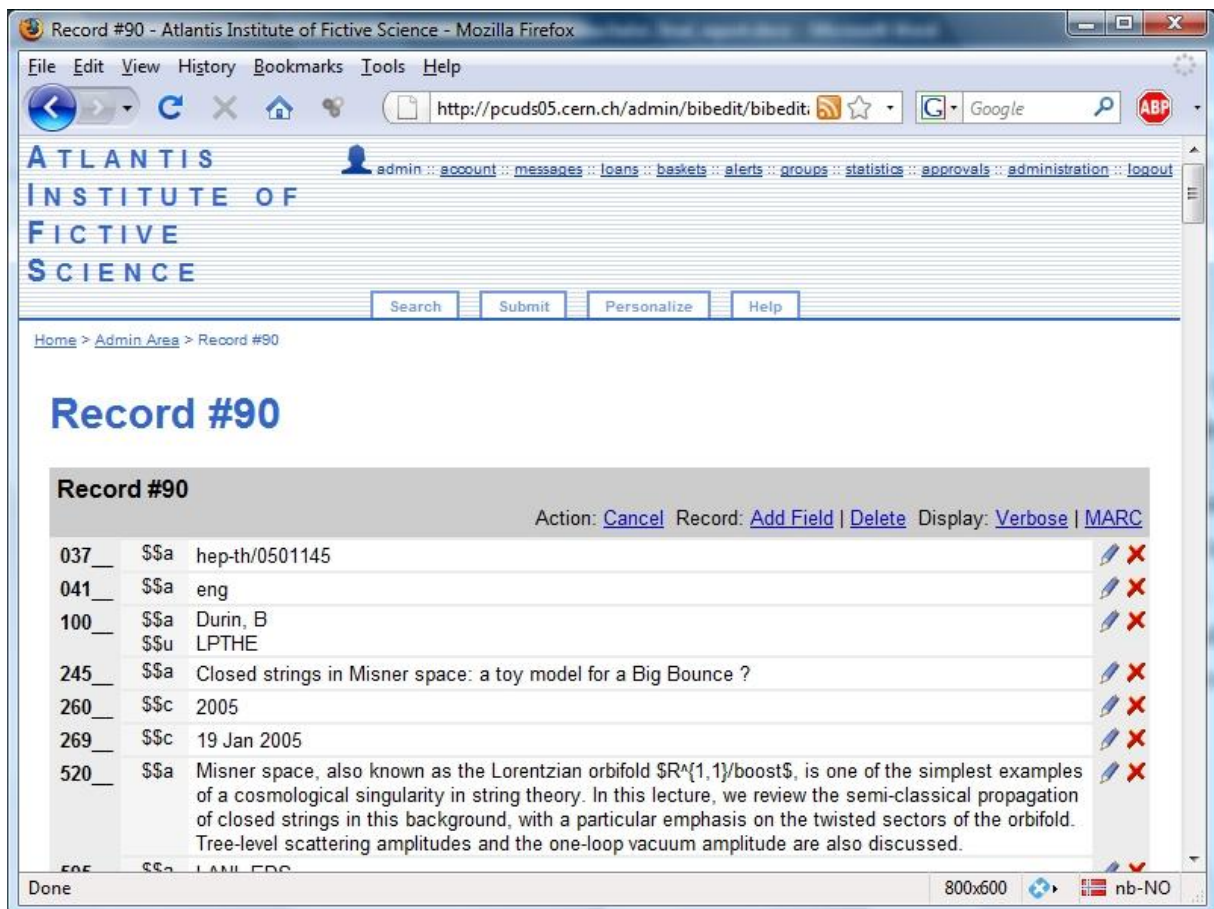


Figure 9: The existing version of the BibEdit tool

One option is to keep developing the existing tool and try to expand it to meet the needs of INSPIRE.

### 3.2.2 The Ajax web tool

Another option is to make use of Ajax technology to attempt to make BibEdit into a more dynamic and responsive web tool. Ajax is an abbreviation for Asynchronous JavaScript and XML;

- Asynchronous: requests can be sent to and received from the server asynchronously, which means that they don't block user interaction with the webpage.
- JavaScript: the clientside needs some extra intelligence to perform these requests. This is taken care of by JavaScript. Thus an important factor to take into consideration is that an Ajax page *requires* JavaScript. It simply won't work as intended if, for example, a user with security concerns has disabled JavaScript.
- XML: XML can be used as the protocol for transferring data between the client and the server, but this isn't set in stone. One could choose another format, like pure text or JSON, and still claim to have an Ajax application. The user wouldn't notice the difference anyway.

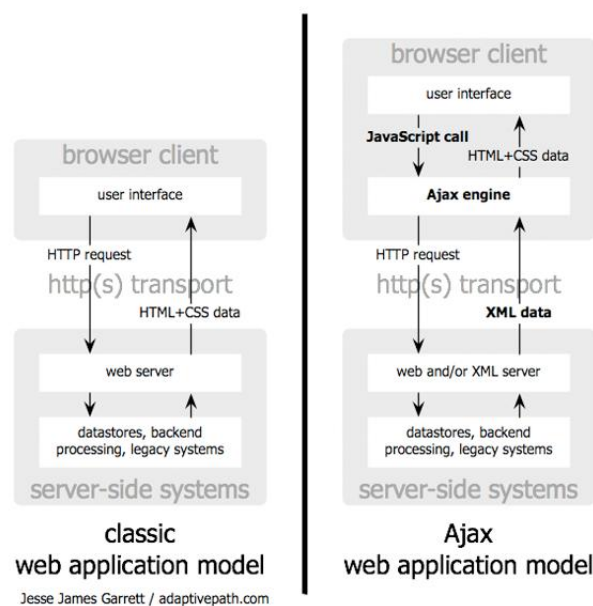


Figure 10: The traditional model for web applications (left) compared to the Ajax model (right)

Making BibEdit into an Ajax application means that the client can communicate with the server without requiring a page reload. Just as importantly it implies that we *require* our users to have JavaScript enabled in their browsers, and once this assumption is made a whole array of new possibilities become available; JavaScript gives great control over a webpage's content and enables it to react to user interaction much more like a desktop application would. Recently there have been revolutionary developments in the performance of the major browsers' integrated JavaScript interpreters. This allows for drawing and manipulating large sections of HTML in relatively short time and enables UI effects like drag and drop or animations.

Still JavaScript is not as fast as optimized Python running on a powerful server, so with this solution it should be carefully considered how much of the serverside processing to move from the server to the client.

### 3.2.3 The desktop application

A third option is to move BibEdit off the web and make it into a desktop application, a concept which requires no further introduction. It would need to be written in some cross-platform language like Python, Java or C++. To be consistent and utilize some of the know-how already present in the CDS section it might be natural to keep it in Python.

Since BibEdit is to be deployed at multiple locations we would have to choose some network protocol to talk to the INSPIRE server. It could of course continue to use HTTP. On the serverside it would either need a backend engine to take care of the interaction with the other modules or, less likely, the other modules would need to get their own interfaces extended to be able to talk directly with the BibEdit application.

### 3.2.4 Discussion of the options

Let's consider the first option; to keep developing the existing web tool. This is the best way to make use of the development that has already been done. It is also the solution that should be the easiest to maintain and develop further by developers already familiar with other parts of the CDS Invenio system, since it keeps BibEdit consistent with its sister modules. So what are the disadvantages of this approach?

Fig. 10 shows the BibEdit tool that already exists in CDS Invenio. The main problem with this tool is that it's built like a classic static web interface, using forms and links to request serverside processing, with the server delivering a brand new page for every click. This leads to a lot of clicking and constant page loads, with the experience being more like surfing a web site then using a powerful editor.

BibEdit will be a central tool in the work of a large number of catalogers. Using BibEdit these catalogers will take care of the editing, enrichment, correction and verification of a great number of records every day. Therefore BibEdit must be developed into a high performance, reliable and robust tool. It is highly doubtful that this objective is achievable using traditional web technology, simply because it was never meant to support rich applications like BibEdit, which leads us to the next option.

Ajax is a set of technologies which helps web applications close some of the gap between them and desktop applications. Instead of having to wait for the server to deliver a new page (and make the user wait) for every action, the Ajax engine lets the client communicate with the server in the background, while the user can keep working. Additionally, assuming that JavaScript is available enables a wide range of possibilities that can help create a better user interface.

The question is of course: Is it enough? Though receiving a lot of attention and huge usage lately, Ajax is still basically all about (ab)using the *XMLHttpRequest*-mechanism to build applications on the web. It's not what Tim Berners-Lee had in mind when he invented the web. In other words; it's just a big hack – a hack with significant browser compatibility issues. How difficult will it be to create a usable product based on this? It can be done, that's for sure. We've all seen Google's recent accomplishments in enabling nothing less than entire Office suites, with word processor, spreadsheet, presentation software and all, on the web. However, we're not Google and must make do with limited resources (me) and time.

The desktop application seems in many ways to be the safest bet, letting you use advanced graphical libraries to create a powerful, lucid and high performance user interface. The network interface appears trickier, but that might be for my lack of experience. The biggest problem is maybe that this would be a major departure from the rest of the CDS Invenio application portfolio, which are either pure web applications or, as some administrative tools, command line tools. It would probably lead to a new, large group of dependencies and make CDS Invenio into an even more complex product than it already is.

The Ajax application stands out as the most interesting alternative. It keeps BibEdit a web application, ensuring immediate global availability without more requirements than an Internet connection and a web browser. The only questions is if we can do it and if it will work well enough to be accepted, preferably even liked, by the catalogers. A natural choice is to make a prototype as quickly as possible and then reevaluate, keeping the desktop application option alive as a fallback plan.



### 3.3 Evaluation and selection of tools and programming languages

BibEdit is one of many modules in the CDS Invenio software package, which has been under development for years. Naturally, most of the tools and programming languages being used have therefore already been selected on a system wide basis. The advocacy and reasoning behind many of the choices are presented on the team's Twiki pages.

Because BibEdit is being completely redone and will be the first module relying heavily on Ajax in CDS Invenio, some technology choices were made after researching, discussing and reaching agreement with the CDS section and the INSPIRE project. This technology will be default for other modules adopting Ajax wholly or partly.

The choices left for me to make alone are rather few, mostly concerning personal tools like which IDE/editor or Linux distribution to use.

I will attempt to list all and describe some of the choices here, also the ones I didn't have any part in making, along with the evaluation leading to them being made.

#### 3.3.1 Technologies – overview

##### **CDS Invenio**

Platform:	Linux (see chapter 3.2.2)
Web server:	Apache (see chapter 3.2.2)
Database server:	MySQL (see chapter 3.2.2)
Server programming:	Python (see chapter 3.2.2)
Apache/Python integration:	Mod_python
Client programming:	JavaScript
Development tools:	GNU Development Tools
Python code quality:	Pylint
Source code management:	Git (see chapter 3.2.3)

##### **Project technology choices**

JavaScript framework:	jQuery (see chapter 3.2.4)
Data transfer protocol:	JSON (see chapter 3.2.5)

##### **Personal technology choices**

Linux distribution:	Ubuntu
Web browser:	Firefox 3.0
Debugging/inspection:	Firebug
Virtualization:	VirtualBox
Code editor:	Emacs (see chapter 3.2.6)
Python Emacs mode:	Standard
JavaScript Emacs mode:	js2-mode
JavaScript compression:	JSTMin
JavaScript code quality:	JSLint

### 3.3.2 CDS Invenio and the LAMP software bundle

CDS Invenio uses the lamp software bundle. The LAMP acronym refers to a combination of software products that together define an application server infrastructure:

- Linux
- Apache
- MySQL
- Python (or some other scripting language like Perl or PHP)

Though these programs weren't designed to work specifically with each other, the combination has become popular because of the ubiquity and low cost of acquisition of the products, all being open sourced and bundled with most current Linux distributions [14]. The open source licensing is of course a prerequisite for any component to be used in a GPL licensed product like CDS Invenio.

The CDS Invenio Twiki talks about the choice of Python as programming language and says that «Python is [a] highly dynamic language with many redefinition capabilities, very well suited to test-driven development and rapid prototyping» [15]. It also quotes several leading computer scientists promoting the qualities of Python, such as Eric Raymond who says about his first experience with the language «To say I was astonished would have been positively wallowing in understatement ... this is an amazing testament to Python's clarity and elegance of design».

### 3.3.3 Source code management system: Git

Git is a free software source code management (SCM) project with an emphasis on being fast. It was initially created by Linus Torvalds for Linux kernel development. Unlike traditional SCM systems it is a distributed system, where every Git working directory is a full-fledged repository with complete history and full revision tracking capabilities, not dependent on network access or a central server [17].

The CDS Invenio project switched from CVS to Git in the end of 2008. Git is thoroughly compared with other options on the CDS Invenio Twiki, perhaps because CERN centrally has chosen to standardize on Subversion as SCM system, so the departure from this standard needs to be justified (though CDS intends to mirror to the Subversion service). The critic against Subversion is roughly the same as against CVS; seemingly simple or commonly used operations, such as file renaming or merging, turns out to be strikingly difficult when using these centralized systems [18].

Git is also compared to other SCM systems using a distributed paradigm, such as Mercurial and Bazaar, however Git wins out as the most powerful and feature rich solution and the one with the most impressive adoption history, being chosen as the preferred solution by numerous high profile software projects such as the Linux Kernel, GNOME, Qt, OLPC and Android [17].

It is however noted that Git is a complex solution and might have a steep learning curve, something I can very much attest to. A blogger talks about Git «...being a tool for designing VCS workflows, as much as being a VCS system itself» [19]. A good understanding of Git really requires a dedicated study. It is not something you will possess after a simple introduction or by trying to use it (a deeply frustrating experience if you don't understand what's going on). After several months I'm now starting to reach a level where I can utilize some of the power of Git, instead of just feeling crippled (and angry). However I still always back up all my work before doing any major Git operations...

### 3.3.4 JavaScript framework: jQuery

A JavaScript framework provides an extra layer of abstraction between the programmer and pure JavaScript code. It simplifies many common operations and, most importantly, relieves you of having to take all kinds of browser incompatibilities, hacks and quirks into consideration when you write your code. The framework takes care of checking for the existence of used features and deploys required workarounds if necessary. Most frameworks also contain powerful mini programs (built-in or available as plugins) to let you create widgets, animations or common user interface controls.

There are at least a dozen high quality JavaScript frameworks available, and at the time I started developing BibEdit no choice had been made as for which to use. Some developers favored jQuery, which already stood out in the crowd because of its adoption with companies like Google and Dell. Shortly after we landed on jQuery, Microsoft and Nokia also announced that they would use jQuery; Microsoft in Visual Studio and their ASP.NET frameworks, Nokia as part of their Web Run-Time platform [19].

jQuery is a lightweight library, with a simple and user-friendly syntax, yet very fast and powerful. It supports a wide range of element selectors, event handling, effects and animations, CSS manipulation and Ajax. It is easily extended with additional plugins and a separate customizable UI package contains popular UI effects and widgets.

Compared to other libraries it came out stronger because of its solid customer base, its simplicity and its small default footprint. The two last factors are important as jQuery might be used for creating small JavaScript programs to enrich existing web pages. This requires that the library is small enough to be included everywhere without noticeably delaying page loading and that it can be used by developers without much JavaScript experience.

### 3.3.5 Data transfer protocol: JSON

JSON is short for JavaScript Object Notation. It is a text-based, human-readable data format for representing simple data structures and associative arrays (called objects) [20]. Fig. 12 – 14 explains the basic JSON data types; objects, arrays and value.

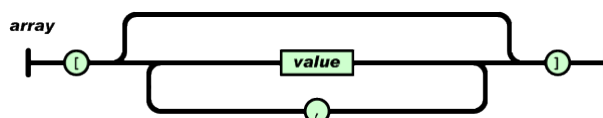


Figure 11: JSON array (from json.org)

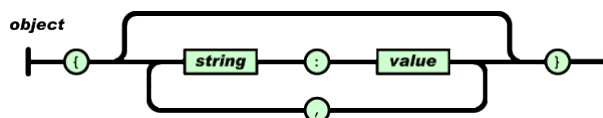


Figure 12: JSON object (from json.org)

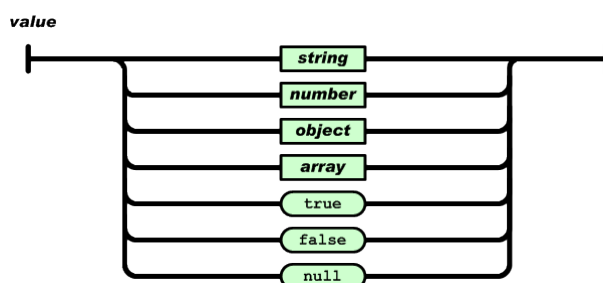


Figure 13: JSON value (from json.org)

The above pictures are taken from json.org, where the full specification can be found, though it's really not much more than what can be understood from these three pictures. JSON is a very simple format, it is uncompressed and lacks the power of XML, however it works perfectly fine for most small to moderately large Ajax applications which don't need to transfer enormous amounts of complex data.

Another advantage of JSON is that it is valid JavaScript and can be interpreted directly on reception (although that is an unsafe way to go about it). It is also *very close* to being valid Python, so only some syntactic parsing is necessary to translate Python data structures into JSON or vice versa, conceptually they are almost the same. So although Invenio is using XML as its internal data storage format, the library which manipulates the record during editing (called BibRecord) represents the record using a Python dictionary – very similar to a JSON object.

I have little experience with XML and of course JSON, simple as it is, made immediate sense to me. I got green light to start prototyping with JSON and it has since worked out very well for my project. There are discussions about writing JSON interfaces for other Invenio modules, so the protocol seems to be gaining some traction with other developers as well.

### 3.3.6 Code editor: Emacs

Though I had a little previous experience with Emacs, I've usually preferred heavier IDEs like Eclipse or Visual Studio and initially I tried adapting Eclipse to the development I was doing. This worked reasonably well with Python, but once I started doing JavaScript development Eclipse failed to deliver, so when I had to set up a new computer anyway I switched to Emacs.

Emacs has some avid fans in the CDS section. The CDS Twiki has this to say about starting to use Emacs: «As a beginner you start using it simply as a “yet another text editor”, and after a while, as the time passes by and your editing habits/needs/expectations progressively grow, you can keep adding more and more of the advanced features to your repertoire, to extend the Emacs functionality itself, to easily adapt it to your (future) needs» [21]. Personally I still think I'm at the beginner level, where Emacs is doing a decent job as code editor, but I don't yet feel like the Jedi hacker using Emacs as his light saber.

For Python development I use the Python mode which comes with Emacs in version 21 or above. For JavaScript development I use the mostly excellent js2-mode [22].

## 4. Design

### 4.1 Architecture

Fig. 15 shows the architecture of the BibEdit application somewhat simplified. The engine running serverside utilizes the BibRecord module to manipulate a bibliographic record and BibUpload to submit a changed record back to the database (first converting it to XML). While the record is being edited it is cached on file using Python's pickle module.

The BibEdit engine also constructs the initial BibEdit page and its menu, with links to the JavaScript files, including the BibEdit scripts controlling user interaction and support libraries like jQuery, JSON and more. Finally it handles any command line interaction with the user.

On the clientside the JavaScript engine handles construction of the record table and user interaction. Every update or action is sent to the server as an Ajax request. The data itself is encoded as JSON. Serverside decoding and encoding of JSON is handled by the Python simplejson library.

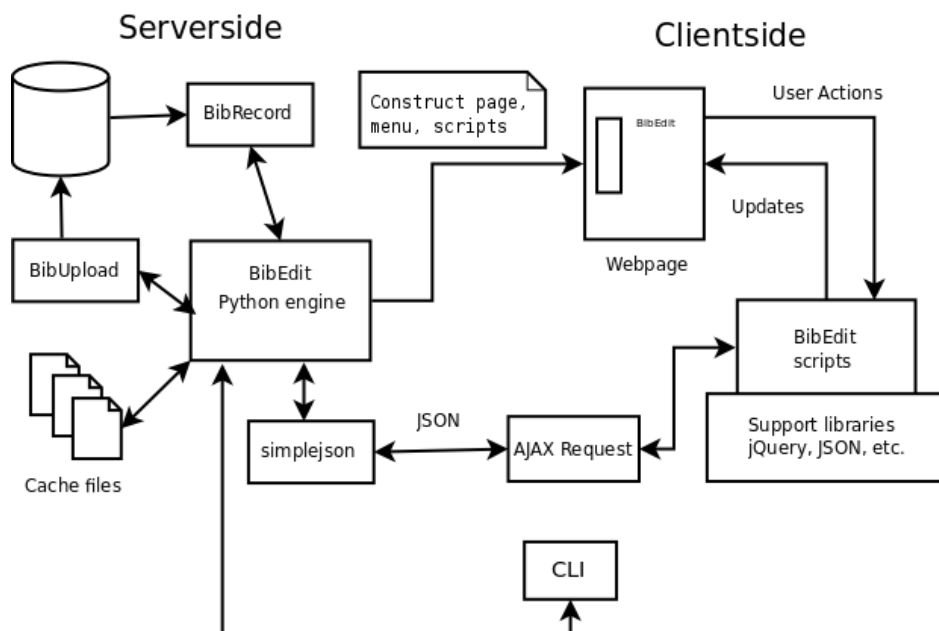


Figure 14: The BibEdit architecture

Fig. 16 gives a more detailed look at the server side Python engine. Most of the files are typical for Invenio modules, such as the webinterface-, templates-, config- and dblayer-modules. A lot of support functionality has been moved from the engine-module to a more neutral utils-module, because of its reusability potential with other modules.

### BibEdit Serverside Python Engine

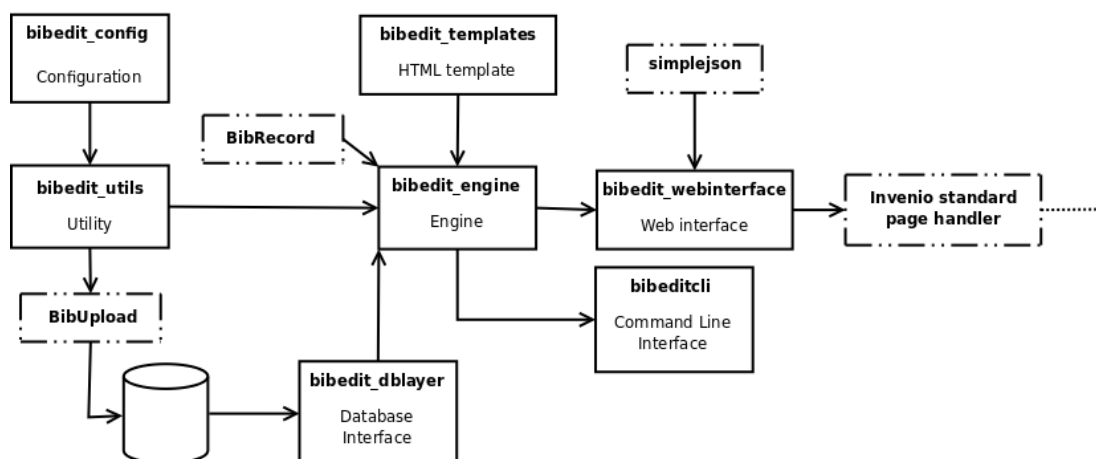


Figure 15: The BibEdit serverside Python engine

Fig. 17 shows the finer points of the client side JavaScript engine. Most core functionality is divided between the *engine*- and the *menu*-script, mainly to avoid the engine growing into a gigantic do-everything script. The engine does the script initialization and sends AJAX requests when necessary. The *display* script generates HTML and the *keys* script is solely event handler functions for hotkeys.

### BibEdit Clientside JavaScript Engine

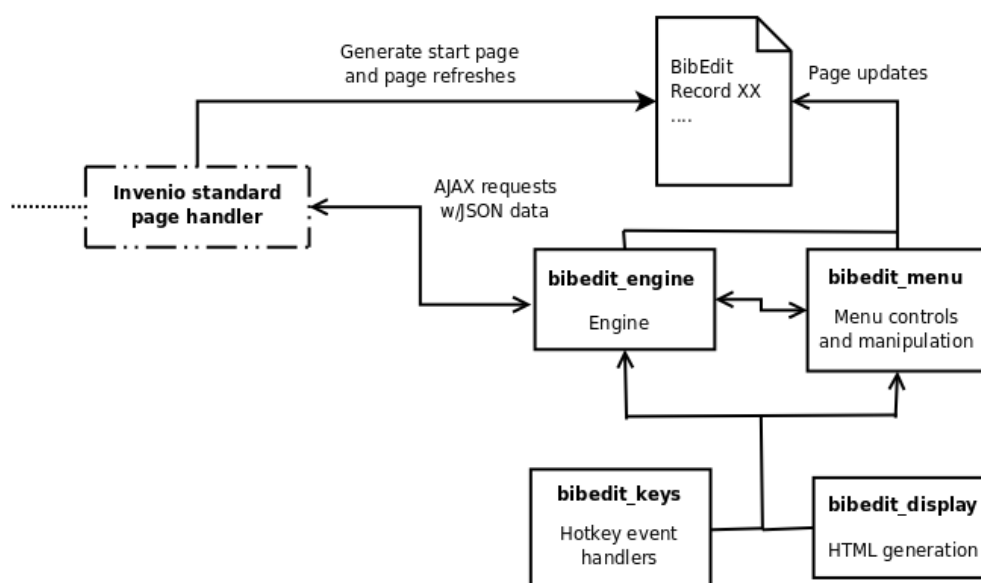


Figure 16: The BibEdit clientside JavaScript engine

## 4.2 Design solutions

### 4.2.1 Browser compatibility

BibEdit will be an administrative backend web tool used by a limited number of catalogers, as opposed to frontend tools available to the masses. Therefore ultimate browser compatibility is not an important goal. In the worst case we can say that BibEdit is only guaranteed to work in one specific major browser and then have all the catalogers standardize on this browser. Still it is preferable if the application is tested and works with the later versions of all major browsers. This isn't such a huge effort as it once were, since browsers are getting better at following agreed upon web standards and since tools like jQuery makes it easier to write browser independent code.

BibEdit releases will be tested against Internet Explorer 7 and 8, Firefox 2 and 3 and the latest public version of Safari, Opera, Chrome and Iceape (Mozilla). Major flaws and bugs will have to be debugged and fixed. This might be a good thing as many problems will surely stem from non-standard code being accepted by some, but not all browsers, rather than browsers not following web standards.

### 4.2.2 URL and navigation

The application is reachable through two types of URLs:

1. Generic: `<SITE_URL>/record/edit/`
2. Record specific: `<SITE_URL>/record/<RECID>/edit/`

It won't make a difference if the final slash is left out.

BibEdit is built to operate without any page reloads at all, giving users an interface reminiscent of a desktop application. However, this requires BibEdit to use a generic URL, not a URL that is changing depending on the state of the application (like which record is being edited etc.) The reason for this is that any URL change will also make the page reload in all modern browsers, defeating the original purpose [23].

Therefore, when being called with a non-generic URL, like `/record/50/edit/`, BibEdit will redirect to a generic URL like `/record/edit/#state=edit&recid=50`. The last URL might not seem more generic then the first, but this is actually a trick. The fragment part of the URL (what comes after the `#` - the hash) can be freely updated by the script, to keep track of state, without causing page reloads.

If correctly implemented, this allows the user to use the browser navigation buttons (*Back*, *Forward*, *Reload*) with expected results in most browsers. Internet Explorer up to version 7 doesn't write changes of the fragment to the page history, so the *Back* button takes you *all* the way back, but this is fixed in IE8 and already works in most other browsers.

This also facilitates calling BibEdit with custom arguments, like some action to perform, some particular view or filter to use etc.



### 4.2.3 Diff and merge

The diff and merge functionality so far hasn't come further than being an interesting design discussion. It is also decided that, when implemented, the main part of this functionality will be part of the BibUpload module. Yet it will also lead to changes in BibEdit and it is in the context of BibEdit it has been discussed.

The root of the problem is that CDS Invenio is a multiuser system, where several users could wish to modify the same record at the same time. How can one avoid conflicts or loss of data as a result of changes being done in parallel? The most obvious example of this is that two catalogers edit the same record at the same time.

The solution to this problem has been to lock records if they are being edited or if they are in the queue to be uploaded, but this strategy has its own problems, such as obstructing the cataloging workflow, unintentional overwrites and costly and complex lock checking.

Therefore a more CVS style solution has been proposed where a changed record will be diffed against the original, so that only the changed fields need to be submitted for uploading. By doing this diff in BibEdit one can submit two streams, an append stream and a change stream, instead of replacing the full record like it is done today. It would also make it possible to catch and alert the cataloger to potential conflicts at an earliest possible stage.

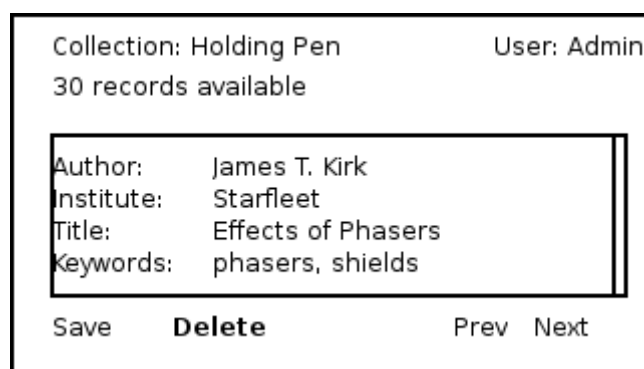
If changes happen in parallel and there is no overlapping between the fieldsets being changed, the changes can easily be merged into the record. If there are conflicts that can't be solved without human intervention, the record should be returned to the cataloger together with the proposed changes so that a manual merge can be performed.

However, adding to the complexity of the problem is the fact that not all modifications need come from BibEdit, they could be batch jobs launched by the MultiEdit module, by automatic formatting or correction tools or by a sysadmin. Therefore another level of diffing and a central authority for merging changes into existing content is needed and this will be placed in BibUpload, which is the gateway for any incoming modifications and the only place where one is guaranteed to catch all conflicts.

## 4.3 Graphical User Interface

### 4.3.1 Mock-up screenshots – early design stage

These mock-ups were created by a colleague in the very early design stage, before I was assigned the project. Fig. 18 shows browsing the records in the holding pen. Fig. 19 shows display of suggested institute names retrieved from an authority record.

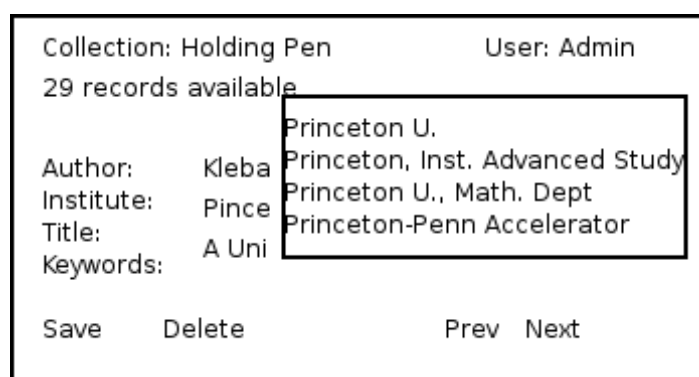


Collection: Holding Pen User: Admin  
30 records available

Author:	James T. Kirk
Institute:	Starfleet
Title:	Effects of Phasers
Keywords:	phasers, shields

Save **Delete** Prev Next

Figure 17: First record of the set shown



Collection: Holding Pen User: Admin  
29 records available

Author:	Kleba
Institute:	Pince
Title:	A Uni
Keywords:	

Princeton U.  
Princeton, Inst. Advanced Study  
Princeton U., Math. Dept  
Princeton-Penn Accelerator

Save Delete Prev Next

Figure 18: Correcting an institute name

### 4.3.2 Mock-up screenshots – advanced design stage

I created these mock-ups using a lot of hand-coded HTML and the frame of the existing INSPIRE alpha version. The pictures attempt to show how the new functionality with basic controls could be fitted into a user interface inspired by the existing BibEdit version.

Fig. 20 shows the basic editing mode with controls for some fundamental operations: adding new field, adding new subfields (the green pluses on the right hand side), deleting fields and subfields (the red X'es) and handles for dragging subfields around (the blue rectangles). It is assumed that content can be edited simply by clicking on it, more on this later.

The screenshot shows the INSPIRE alpha editor interface. At the top is the INSPIRE alpha logo. Below it is a navigation bar with links: EDITOR, BibCHECK, HISTORY, MULTIEDIT, and SETTINGS. The main content area displays 'Record: 35' and 'Format: MARC' with buttons for 'Go to record', 'Submit', 'Cancel', and 'Delete'. Below this, it says 'Displaying record #35' and 'Current version (modified)'. The main table shows the record details:

Add new field			
001			35
003			SzGeCERN
260	—	\$\$\$a	Jyvaskyla
		\$\$\$b	Finland Univ. Dept. Phys.
		\$\$\$c	Jul 1982
909	C0	\$\$\$y	1982

Figure 19: The editor

Fig. 21 shows the BibCheck functionality and is perhaps the most interesting. The idea of browsing the records in the holding pen is kept and a *Submit and Next* button takes the cataloger to the next record he/she should work on.

The results from BibCheck are of three types; automatic changes, suggested changes and warnings, with controls to let the cataloger browser through them. Automatic changes are shown with attached information about what has been done, and perhaps which rule caused it to happen. Suggested changes can be accepted or rejected by the cataloger, while warnings need to be manually resolved.

IN SPIRE  
alpha

EDITOR :: **BibCHECK** :: HISTORY :: MULTIEDIT :: SETTINGS

[Home](#) > [BibEdit](#)

Record: 35  
[Go to record](#)

Format: MARC [Verbose](#)

[Run BibCheck](#) [Submit and Next](#)  
[Accept all changes](#) [<<Previous](#) [Next>>](#)  
[Decline all changes](#)

**Displaying record #35**  
Current version (modified)

**BibCheck status:**  
1 automatic change 1 suggested change 2 warnings

				<a href="#">Add new field</a>
✖	001			35
✖	003			SzGeCERN
✖	088	\$\$a	✖	Foo-BaBar-2002-10
✖	123	\$\$a	✖	LHC <a href="#">Large Hadron Collider</a>
✖	260	\$\$a	✖	Jyvaskyla
		\$\$b	✖	Finland Univ. Dept. Phys.
		\$\$c	✖	Jul 1982
		\$\$q	✖	Hey, how are you?
				⚠ Subfield 260 \$\$q is not among allowed ones [abc]
				⚠ Mandatory subfield 710 \$\$a is missing
✖	909	C0	\$\$y	1982

Figure 20: BibCheck mode

Fig. 22 shows the History functionality with controls for selecting revisions, comparing revisions and reverting to a specific revision of the record.

Home > BibEdit

Record: 35 [Go to record](#)

Format: [MARC](#) [Verbose](#)

View revision: 35.20080808133030 [View](#)

Compare revisions: <Choose revision> <Choose revision> [Compare](#)

Revert to revision: <Choose revision> [Revert](#)

**Displaying record #35**  
Current version (modified)

+ Add new field				
001				35
003				SzGeCERN
260	—	\$\$a		Jyväskylä
		\$\$b		Finland Univ. Dept. Phys.
		\$\$c		Jul 1982
909	C0	\$\$y		1982

Figure 21: History mode

Fig. 23 shows how the autosuggest mechanism could work for fields with lookups in authority records, in this case an institute.

001				35
003				SzGeCERN
260	—	\$\$a		Jyväskylä
		\$\$b		Fin
		\$\$c		Finland Dept. Univ. Phys.
				Finland Dept. Univ. Math.
				Finland Helsinki Physics Center
				Finmarksvidden Physics Group
				Finstat Univ. Dep. Phys.
				Finstat Imperial College

[Close](#)

Figure 22: Autosuggest

### 4.3.3 GUI layout

BibEdit is using Invenio's standard page handler. The page handler interprets the URL of requests and forwards them to the corresponding modules. Usually these modules then generate some content which it returns to the page handler. The page handler then inserts this content into the frame of the Invenio system, adding headers, footers, menu and navigation to the page (responses to Ajax requests are treated differently of course).

This means that the application BibEdit is set and operates in the same frame as all other Invenio modules. This is good because it gives a consistent and professional look across modules. It is bad because it places some serious restrictions on the GUI. For example; if elements are positioned in a way that removes them from the *flow* of the page, care must be taken so they don't crash into the static header and footer.

Since the application will display records that in most cases will be too large to fit on one screen, the rest of the record will be available by scrolling vertically. Still we want the user to have the controls at hand at any given time, leaving a dynamically moved sidebar menu as the better option. It is most natural to fit it on the left side of the page, leaving us with a basic GUI structure like this:

#### BibEdit GUI Structure

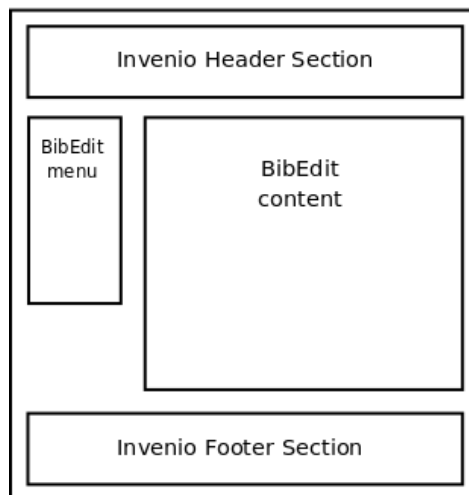


Figure 23: The basic structure of the BibEdit GUI

Fig. 24 was developed at a later stage and is therefore different from the design sketches in chapter 4.3.2, mainly with regards to the placement of the menu.

#### 4.3.4 In-place editing

Fig. 25 shows a mock-up of how in-place editing could look. The basic idea is that to avoid breaking the flow of the user by bringing up some dialog or, worse, taking the user to another page to modify some content; when the user indicates he/she wants to edit some content, we replace the static content with a form where the user can modify the text. With the power of JavaScript it shouldn't be an insurmountable task. To eliminate the need for a set of *edit*-buttons for every possible content section, editing could be initiated simply by double clicking at the part you would like to edit.

This model could be extended to inserting new as well as modifying existing content. New fields or subfields could be placed directly into the table.

✖	001			35
✖	003			SzGeCERN
✖	260	—	\$\$a	Jyvaskyla
			\$\$b	Finland Univ. Dept. Phys.
			\$\$c	Jul 1982
✖	909	C0	\$\$y	1982

Figure 24: In-place editing

There are many jQuery plugins for this kind of functionality. Hopefully we can find one that fits our style of use and avoid writing something from scratch.

#### 4.3.5 Optimizing GUI performance

Because BibEdit is building and manipulating a possibly large and complex web page depending on the size of the record in question and because the main selling point of building an Ajax application is a faster GUI, it is critical that the performance of the GUI is experienced to be somewhere between good and great.

A simple trick to make the page load faster is to build all the HTML content using a few simple functions that concatenates it all together into one big string, before setting it as the *inner HTML* of some DIV. This is an extremely time saving option compared to using JavaScript DOM manipulation to create and structure the different elements together.

To maximize the responsiveness of the GUI any update is assumed to be successful serverside. This means that the JavaScript, which contains a full copy of the record structure, can immediately do the necessary processing and update the view independently of the Ajax request which is sent to the server in the background. The server updates its copy of the record and then returns the result (success or failure) to the client. This results in an extremely fast GUI since it's practically unaffected by any delay induced by network or server problems.

The problem is of course if something fails on the serverside (which should be a *very* rare event), in which case we have a serious situation where the clients view is out of sync. As soon as this is noticed the user should be blocked from doing any more work until the error that caused the problem has been addressed and the view reloaded. To be completely secure there should also be consistency checks to confirm that the client and serverside record data is in sync, but this could be as easy as comparing some computed checksum at regular intervals.

It is nice to keep in mind that the performance situation should only improve as the browsers JavaScript interpreters are improved and the average clients processing capabilities keeps increasing, so anything that appears fast now should be blistering fast in a couple of years.

#### 4.3.6 Streamlining the GUI

Even with the power of Ajax and modern web standards and browsers, the tools for creating a web GUI is nowhere near as good as those existing for desktop applications. Some ingenuity is needed to squeeze more power out of the possibilities present in today's web standards. What are some of the techniques that could be utilized to boost cataloger efficiency even more?

##### **Hotkeys**

The application should, as much as possible, be able to rely solely on keyboard input. This means that every operation possible with the mouse should have a keyboard equivalent, maybe not as efficient because indicating positions with keyboard is bound to be inefficient, but doable. Experienced catalogers will hopefully learn most or all of these shortcuts and use them to speed up their work by minimizing mouse usage.

##### **Views and filters**

Filters should be implemented to make it possible for the cataloger to restrict his/hers view to the fields of interest. Some records are very large; making it both difficult for the cataloger to find the field he/she is searching for and creating a huge load on the browser which has to draw the full record on the page. Using filters both these problems can be avoided and the cataloger can easily locate the part of the record he/she wants to work on. Filters might be conserved across records making life easier for a cataloger doing some specific maintenance on a large set of records.

More advanced than filters is the idea of having optional views, for example a view where subfields are showed horizontally instead of vertically, something that might be a valuable help for catalogers doing manual proofreading of large field groups. However most ideas such as this serves to simplify cataloging of the major data types; keywords, references and authors – all of which are now having customized editing modes developed. No doubt the implementation of ideal views will be a primary objective in this project.



### **Smart selection modes**

Rectangle selection is native behavior to most desktop applications, not so for web browsers where determining what elements were beneath a drawn rectangle is a job for fairly advanced plugins, and with difficulty increasing proportionally with the complexity of the web page. Other methods of doing selection are needed.

One example is a so called selection mode, which a user might enter by clicking some hotkey. In this mode, by using the mouseover-event fired by elements when the mouse are moved over them, the user can select a large range of fields by just dragging the mouse across them. Another option is a mode where the user selects the start and the end of the selection, thereby automatically selecting all elements in between. Or perhaps the view filters could be useful in this context too, by first specifying a filter and then selecting all fields that match the criteria used.

Of course most importantly there should be plenty of ways to select individual fields and subfields using both the mouse and the keyboard shortcuts.

### **Using the power of text editors**

There is a huge difference between the primitive text editing capabilities of most web applications and the power and rich feature set of advanced text editors such as Emacs or vim, editors that have been extended and developed for the single purpose of efficiently editing text over the last 20-30 years.

Many users would appreciate if they were allowed to edit the whole record or parts of it in such environments. By using plugins such as Firefox's MozEX this is possible. MozEX is an extension which allows the user to use external programs for, among other things, editing the content of textareas [24]. The content of some subfields are so large that it might be interesting to be able to edit it via MozEX. Even more interesting though, is opening and editing the full MARCXML (the XML representation of MARC) of a record in this way. This would, for example, give the user an easy way to copy and paste large pieces of content into one or several records. Native Invenio tools could be used to parse the resulting MARCXML.

### **Undo**

An undo option is an important feature of any editor and has found its place in most advanced Ajax tools on the web today. When there is an easily available undo function, there is less need for confirmation dialog boxes everywhere, which in turn results in smoother user interaction. The undo function need only go one or a few steps back and having a *redo* function is less important. It is however, not obvious how to implement it, since it actually needs to reverse actions with potential side-effects. Smart techniques for creating undo for Ajax applications is discussed at the Humanized blog [25, 26].

## 5. Implementation

I will here discuss interesting features and details in the implementation of BibEdit. For a look at the complete code, please see the files attached on the CD or visit <http://cdsware.cern.ch/repo/> for “bleeding edge” sources.

The code below might be commented more extensively than the real source to clarify what it does to the reader. Also some lines deemed irrelevant to the context have been left out.

### 5.1 Ajax

#### 5.1.1 Ajax requests with jQuery and JSON

jQuery has good support for Ajax – setting it up and sending requests are very easy.

##### Ajax setup:

```
$.ajaxSetup(  
  { cache: false, // Don't use cached data on the client.  
    dataType: 'json', // Decode the response as JSON.  
    error: onReqError, // Use this function on errors.  
    type: 'POST', // Set request type to be POST.  
    url: '/record/edit/' // Send requests to this URL.  
  }  
);
```

##### Ajax request:

```
function createReq(data, onSuccess){  
  /*  
   * Create Ajax request.  
   */  
  $.ajax({  
    data: {  
      jsondata: JSON.stringify(data) // Create a JSON string.  
    },  
    // Call this function if request returns successfully.  
    success: onSuccess(json);  
  })  
};
```

The JSON JavaScript library [28] has a function, *stringify*, for creating a properly escaped string of JSON from a JavaScript variable or data structure.

The *data* parameter for this method might look like this:

```
{ requestType: "getRecord",  
  searchType: "recID",  
  recID: 36 }
```

Its meaning should be obvious: “Get the record with record ID 36”.

### Serverside request handling:

Serverside the presence of JSON data identifies the request as an Ajax request. The JSON is then extracted (not shown below) from the request data to a variable called *jsondata*.

```
import simplejson as json
...
# Wash as string, decode with the simplejson library.
data = json.loads(str(jsondata))
# Deunicode all strings (CDS Invenio doesn't have complete Unicode
# support).
data = json_unicode_to_utf8(data)
...
# Go on extracting data as necessary.
recid = data['recID']
```

In the beginning the JSON format left some bumps in the road resulting in the following “checklist”;

- don't use JSON if it's not necessary, for example when creating JavaScript variables during initial page construction
- be sure that what you send and receive is a string
- be sure to use **double quotes only** when creating Python strings that will be encoded as JSON
- use JavaScript and Python JSON libraries to get things properly escaped and encoded/decoded
- simplejson creates Python Unicode strings, run the data structure through the `json_unicode_to_utf8` method to get regular UTF8 strings

### 5.1.2 The URL state manager

As explained in chapter 4.2.2 we wanted to make BibEdit aware of its current state by updating the fragment part of the URL, the part following the # (hash) symbol, thereby enabling the use of browser navigation buttons such as *Back* and *Refresh* without breaking the application, and at the same time avoiding any unnecessary page loads caused by changing the rest of the URL. An example of such a generic, but stateful URL is:

```
http://<SITE_URL>/record/edit/#state=edit&recid=100
```

This URL signals that you are currently editing record 100.

Manipulating the hash using JavaScript is easy; unfortunately it won't always be the script that modifies the hash. In a modern browser, hash changes are written to page history, so if the user clicks the *Back*-button, the hash will change. We might also have a scenario where the user tries to change the hash himself, for example to fetch another record. Anyhow, this is a problem, because if we don't do anything about it, nothing will happen. The hash is normally used to jump around between anchors on the page. There are no anchors with names like these, so the browser will just ignore the change.

The solution is to implement a hash monitor that regularly checks if the hash has changed and, if it has, induces the expected behavior from the application. In a simple application this can be implemented in a nice and clean way; for any user action, simply change the hash to indicate the new state of the application, let the hash monitor detect the change and launch the corresponding operations.

Unfortunately not so with BibEdit where a change of state can have important side-effects. Imagine if we do some changes to record 100 and then submit it, which will lead to a state like:

```
http://<SITE_URL>/record/edit/#state=submit&recid=100
```

In the model described above, this state implies that we want to submit record 100. If the user now does something else and then starts clicking the *Back*-button until he/she hits this state again, it's pretty certain that the user don't mean to resubmit record 100. He/she would simply expect to see the same confirmation screen that was shown when the record was submitted a while ago.

This leads to a rather complex solution which I've called the URL state manager. Operations being invoked from the application (by clicking buttons etc.) leads to the URL state manager being disabled while the operations are performed and the URL updated. When the URL state manager is enabled it notices any changes to the fragment part of the URL and tries to evaluate if they are valid or not (it could be the user messing around with the URL). If they appear valid (maybe the user clicked the *Back* button) then it handles the change in different ways depending on the new URL. If, for example, the URL implies that we should get another record, it goes ahead and gets it, following more or less the same routine as if the user had requested the record using the ordinary menu controls. If the URL implies that we have entered some state where a record has just been submitted or deleted, for example, it just redisplay the receipt of this to the user.

If the change appears invalid it is simply ignored. To keep itself independent of the URL it has an internal state variable which always reflects the actual state of the application, regardless of what the URL shows.

So in general it tries to be smart about what to do and let the user have the impression that the application understands him/her very well. Of course the implementation is already rather complicated and will be even worse if additional state parameters are added, which they probably will. This mechanism is probably the most complex and least understandable part of *BibEdit*, and it should be evaluated with the objective of finding a better and more flexible implementation.

## 5.2 GUI

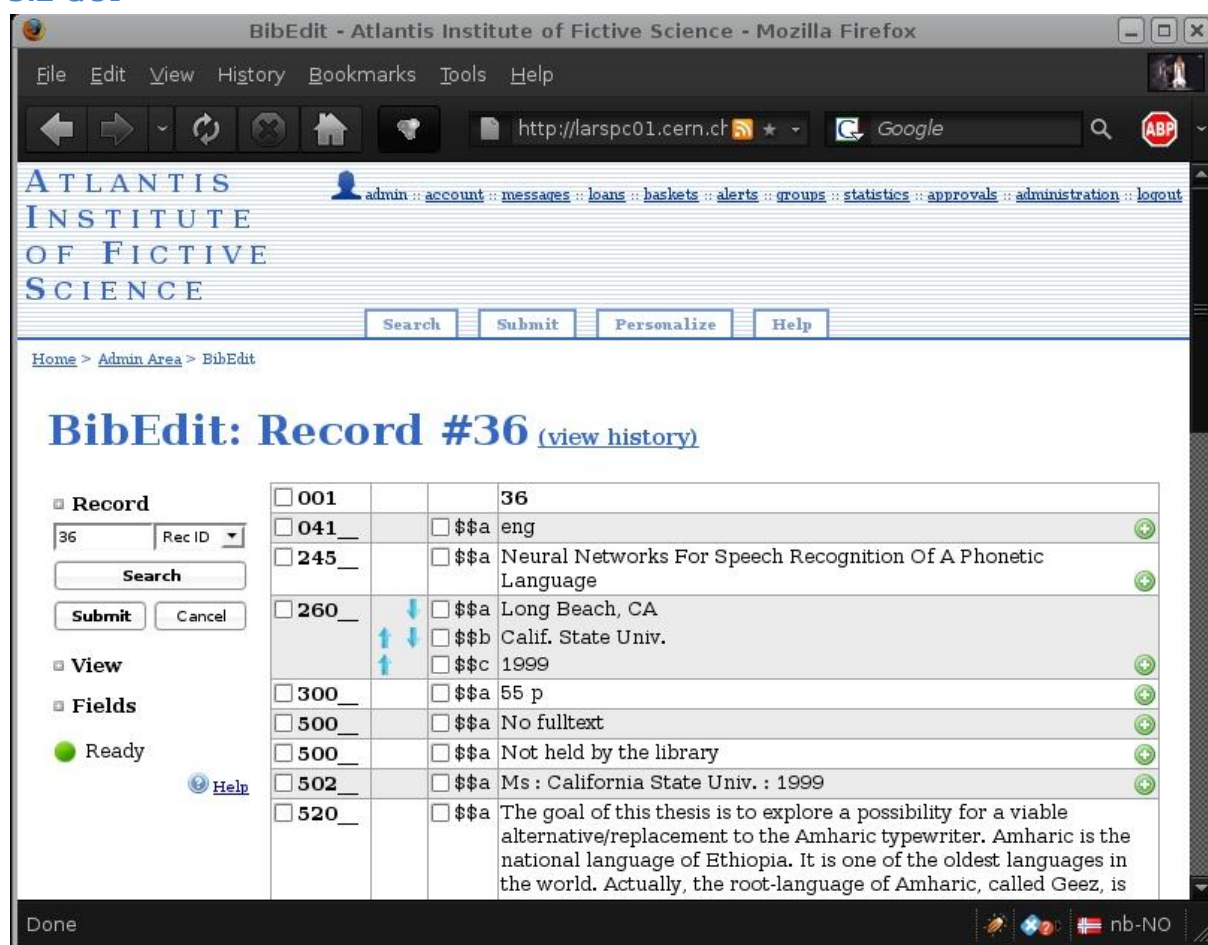


Figure 25: The new BibEdit application

### 5.2.1 HTML, CSS

BibEdit is implemented using standards compliant XHTML 1.1. XHTML 1.1 differs from HTML 4.01 only in a few ways like requiring you to close all tags and use only lowercase element names [27]. It is compliant to the best of the author's ability, as it is hard to validate dynamically generated content against central authorities such as W3C. As all CDS Invenio modules it uses CSS to do nearly all of the element styling. The style rules are found in a central CSS file called *cds.css*.

### 5.2.2 IDs and classes

As BibEdit turned into a heavily complex web application, my approach of naming elements as I was writing code quickly turned into a mess. Though, as most programmers, I always strive for order and consistency, it was clear that some overall rules and structure were necessary in order to not lose control.

Both during the initial design phase and during the refactoring following my realization that things weren't going as well as planned, I tried several approaches to using IDs and classes that in turn all failed, like;

- Assigning multiple IDs. Not very clever and also invalid XHTML.
- Using a combination of IDs and multiple classes in an attempt to make the future implementation of field filters easier. I tried assigning IDs and groups of classes to field elements that would make it easy to filter them given almost any criteria. Of course this got way too complicated really fast.
- Overloading IDs with information by trying to use them both for storing information about the object and functioning as identifier and locator for the element. This made the parsing of element IDs highly complicated and unpredictable, leading me to...
- ...try to apply my own query-like syntax to IDs. This first seemed to work fine and then failed massively in browsers that were trying to be standards compliant. As it turns out the only special characters allowed in IDs are hyphens, underscores, colons and periods.

In the end I landed on the following rules:

**IDs** are used to uniquely identify and locate elements. They are formed using the type of the element and enough information that it can be uniquely and conveniently identified. Underscores separate the different parts of the ID.

The data type is typically a camel-case concatenation of the element type using Hungarian Notation (here button - *btn*) and a description of the role or purpose of the element.

Example template: <data type>\_<tag>\_<field index>\_<subfield index>

Example ID: `btnMoveSubfieldUp_270_6_1`

**Classes** are used to locate groups of elements for selection or for styling with CSS.

Classes are named more freely, but always in camel-case and starting with *bibEdit* to separate them from other modules' classes.

Examples:

`bibEditTxtSubfieldCode`

`bibEditInputError`

### 5.2.3 The left-side menu

#### Positioning

The left-side menu should always be available to the user, even when he scrolls down through the record. The obvious way of doing this is by giving it a fixed position on the screen using CSS' *position: fixed*, but this is not optimal. Ideally you would want the menu to be placed towards the top of the left side, as anything else would look strange. The problem is that at the top of the record, this space is occupied by the header. How can this be solved? By using JavaScript, it's actually not that difficult:

```
function positionMenu(){
  /*
   * Dynamically position menu based on vertical scroll distance.
   */
  var newYscroll = $(document).scrollTop();
  // Only care if there has been some major scrolling.
  if (Math.abs(newYscroll - positionMenu.yScroll) > 10){
    // If scroll distance is less then 200px, position menu in sufficient
    // distance from header.
    if (newYscroll < 200)
      $('#bibEditMenu').animate({
        'top': 220 - newYscroll}, 'fast');
    // If scroll distance has crossed 200px, fix menu 50px from top.
    else if (positionMenu.yScroll < 200 && newYscroll > 200)
      $('#bibEditMenu').animate({
        'top': 50}, 'fast');
    positionMenu.yScroll = newYscroll;
  }
}
```

This function used fixed positioning to position the menu, but if it's close to the top it also pushes it a bit down to avoid collision with the page header. Notice jQuery's `animate` function, which tries to give the menu a smooth movement as it's being moved. Another cool trick is that the *yScroll*-value is stored in the namespace of the function, which makes sense since it's only being used by this function. This avoids polluting the global namespace with too many variables.



Figure 26: Top of the page - menu pushed down to avoid header section

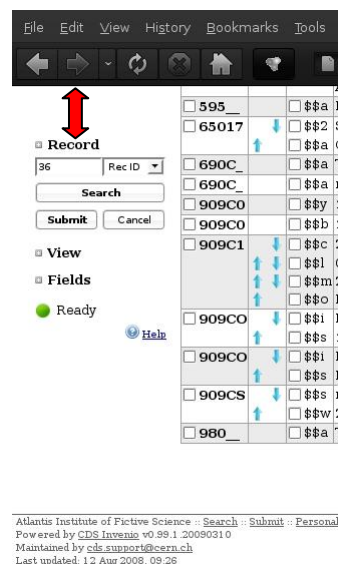


Figure 27: Bottom of the page - menu fixed to upper position

## Menu sections

It didn't come as any surprise that I quickly ran out of menu space for adding additional controls. An accordion-like solution, with expandable/compressible menu sections appeared to be a suitable solution for this sort of vertical menu. The sections are expanded or compressed with a button in the form of a *plus*-symbol. While this should be a familiar solution to most technophiles, some users seem to be a bit lost by it. This isn't a big deal since it's something they will learn quickly and aren't likely to forget, but perhaps the learning process could be eased by finding some even more obvious buttons or adding a little help message.

### BibEdit:



Figure 28: Menu with all sections compressed

### BibEdit:



Figure 29: Menu with all sections expanded



The code for doing this isn't complicated. This event handler is attached to each of the plus-buttons and another event handler doing the opposite to every minus-button.

```
function expandMenuSection() {
  /*
   * Expand a section of the menu.
   */
  var parent = $(this).parent();
  parent.closest('.bibEditMenuSection').find('.bibEditMenuMore').show();
  // Some more code to make sure the menu doesn't expand into the footer of
  // the page and to swap the plus-button with a minus-button...
}
```

Rows that should be shown when the plus-button is clicked are identified by the *bibEditMenuMore*-class.

### Status messages

Though no one actually requested this feature I added it on my own initiative and have since only gotten positive feedback on it and even a request to add more information. I'm talking about the status message being shown at the bottom of the menu which might look something like in figures 31-33.

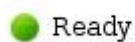


Figure 30: Status message - application ready

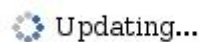


Figure 31: Status message - application working

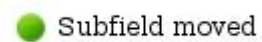


Figure 32: Status message - confirming successful action

In an Ajax application like this a lot of stuff is happening all the time, but since the Ajax requests are asynchronous the interface won't be locked by some hourglass symbol, which kind of leaves you in the dark about what's really going on. Some people might be curious as to what is actually happening behind the curtains, which these status messages will give you a rough idea about. If it says "Updating..." something is actually going on (though this message is rarely seen for more than a second at the time) and if it confirms some action, you should be able to feel confident that the action actually succeeded server side and that the request has returned successfully.

You might think that most users really aren't interested or are only getting confused by these messages, and you might be right, but so far they seem to be an appreciated feature rather than a nuisance. Of course they shouldn't be too detailed, it's not meant to be output for debugging and there is always the danger that they, in effect, takes the "asynchronous" out of Ajax, with users patiently waiting for them to return to "Ready" before they dare continue working. The feature should be evaluated further during user testing and questioning. It provides helpful information during development and testing, but might not be optimal for production use.

### 5.2.4 In-place editing

The in-place editing was an important feature to accomplish the all-functionality-on-one-page vision. The thing about traditional web pages is that either you display static information or, if you want the user to be able to make modifications, you display that information in a form. Now we couldn't show the entire record like one big form, with text input boxes for every piece of editable information, so some more tricks were needed.

And again comes jQuery to the rescue; the in-place editing is implemented with the help of two jQuery plugins.

#### Jeditable

The Jeditable plugin [29] lets the user click some block of text which then becomes a form. The user may then edit the content, save it and watch the form turn into normal text again.

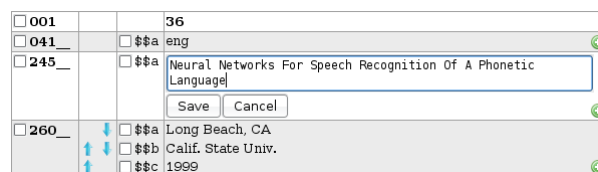


Figure 33: Editing text with Jeditable

Initially the edit mode was entered by just clicking on some content. Later, when realizing that this made selecting text difficult, I changed it to double-clicking (OK, so that makes selecting words more difficult again, but you can't get everything).

To avoid having to add the plugin to every cell with editable content when loading the page, which would give a large performance penalty, I add it on double click and then I programmatically double click the cell again to invoke it. Later, to boost performance even more, I changed from local event handlers for the double click event to just having one global handler attached to the *document* object, which then analyzes which element the mouse pointer was above when you clicked. This works nicely, at least for a rather rare action like double clicking.

Fortunately the Jeditable plugin is easily customizable so it wasn't too difficult to adapt it to my needs. When the user double clicks it needs to get the content from the record data, not from the table where HTML entities are escaped. Then, when the user saves his or hers changes, the text should be processed again, an Ajax request sent, the local record data updated and the table cell redisplayed with the new content (again escaping the HTML). It's not too hard, but I won't show the code here.

## Autogrow

The Autogrow plugin [30] adds support for autogrowing text fields that smoothly expands to fit your text when you write or add new lines, a concept first made popular by Facebook [31]. It is added as a simple argument to Jeditable.

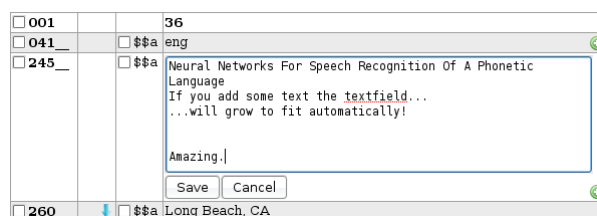


Figure 34: Autogrowing text fields with the Autogrow plugin

### 5.2.5 In-place adding of fields

The adding of fields was originally designed to be done in popup-like dialogs with forms, where the user would fill in the field data. After saving, the data would then be inserted at the correct place in the record. However, after prototyping it became clear that this was breaking the user's flow of work and removing him/her from the main workspace in just the way we wanted to avoid.

So I chose an approach closer to the way in-place editing was implemented, where adding new fields or subfields happens directly in the content. Of course, when adding an empty field, there is no way of determining where it should go in advance, so it's just squeezed in near the top of the table. To make sure there is no confusion the user's browser is also scrolled to this location and the empty form is highlighted so it's sure to attract the attention of the user.

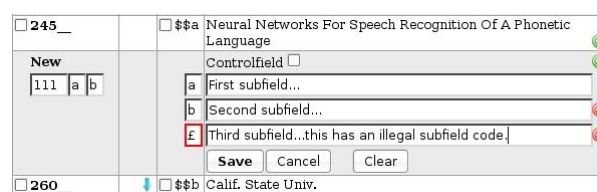


Figure 35: Adding a new field directly in the record

Of course there is a need for validating the users input and alerting him/her to any problems detected. The MARC is checked against the definitions in the standard and any errors are marked by a fat, red border, like the illegal subfield code in figures 36-38. If the user should still try to submit, it will result in one of two possible outcomes. If the error leaves the rest of the input unusable, like in fig. 37 where the only provided subfield code is illegal, it is a critical error that the user must fix before being allowed to save.

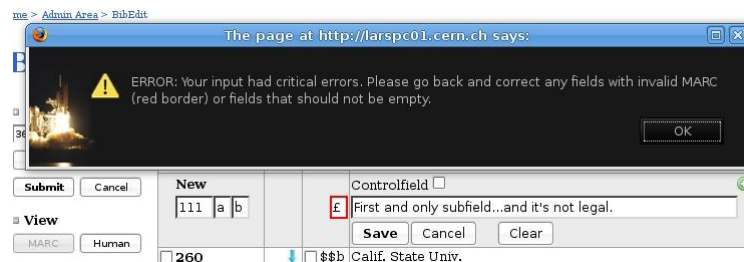


Figure 36: Saving a field with critical errors

If, like in fig. 38, the error only results in one or a few illegal subfields (out of many), the user will receive a warning which he/she may choose to ignore. The field will then be saved, but the illegal subfields will be discarded.

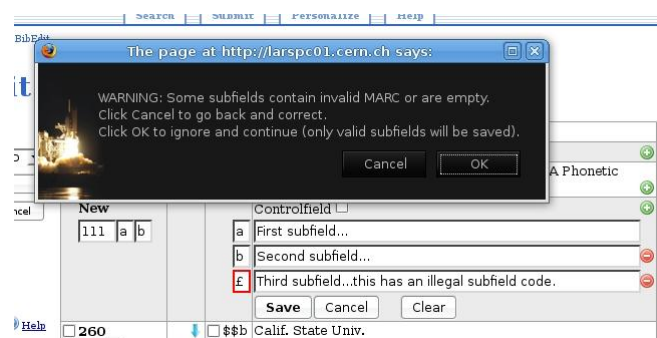


Figure 37: Saving a field with non-critical errors

I have shown the process for adding fields. The process of adding subfields to an existing field are very much the same, so it serves no purpose to demonstrate that as well.

### 5.2.6 Hotkeys

To quote quirksmode.org: «Detecting the user's keystrokes turns out to be a rather specialized branch of event handling...The first problem is that there is no standard for key events» [32]. Different browsers have different interpretations of the three types of keyboard events generated when a key is pressed and alternates between showing *keyCodes* and *charCodes*, representing respectively physical keys and ASCII codes. Add system key combos to the mix and you have a complete mess.

To enable a rich selection of hotkeys for nearly all possible actions BibEdit uses the jQuery Hotkeys Plugin [33]. It turns the otherwise painful job of mapping a keyboard combo to a function, into this:

```
// Add subfield in form.
$(document).bind('keydown', {combi: 'ctrl+shift+e'}, onKeyCtrlShiftE);
```

The above code binds the *keydown* event, if the user clicks *Ctrl+Shift+e*, to the function *onKeyCtrlShiftE*. It even overrides jQuery's own event handler functions, making the code completely interchangeable with standard jQuery.

## 2.1 KEYBOARD SHORTCUTS

### Basic record actions

Shortcut	Definition	Action
g	Select	Go to the record selection field
Shift+s	Submit	Submit the record.
Shift+c	Cancel	Cancel editing of the record.
Shift+d	Delete	Delete the record.
Shift+t	Tags	Toggle MARC/human tags.
a	Add field	Add a new, empty field.
o	Sort fields	Sort fields. This simply updates the view, it doesn't change the record itself.
Del	Delete selected	Deletes all selected fields.

### Focused (clicked) subfield

Shortcut	Definition	Action
Tab	Next	Move focus to next subfield.
Shift+Tab	Previous	Move focus to previous subfield.

Figure 38: A piece of the help file on keyboard shortcuts

To reach the goal of being able to perform all or close to all actions using the keyboard, a way to let the user browse record fields and make selections were needed. The yellow highlighting shown in fig. 40 shows a focused subfield, while the field and the subfields highlighted in blue have been selected and can, for example, be deleted in one operation. Fields receive focus by being clicked or tabbed through. Fields are selected by clicking the select boxes or by hitting *space* or *shift+space* when the subfield or field you want to select has focus.

<input type="checkbox"/> 909C0	<input type="checkbox"/> \$b 14
<input type="checkbox"/> 909C1	<input type="checkbox"/> \$c 2000-09-22
	<input type="checkbox"/> \$l 00
	<input checked="" type="checkbox"/> \$m 2002-02-22
	<input checked="" type="checkbox"/> \$o BATCH
<input checked="" type="checkbox"/> 909C0	<input checked="" type="checkbox"/> \$1 PROQUEST
	<input checked="" type="checkbox"/> \$s 1397120
<input type="checkbox"/> 909C0	<input type="checkbox"/> \$i PROQUEST
	<input type="checkbox"/> \$s MAI36/02p445Apr2000
<input type="checkbox"/> 909CS	<input type="checkbox"/> \$s n

Figure 39: Selection and focusing

## 6. Testing

### 6.1 The users

The users of the finished application are all CDS Invenio users with record curator authorization (catalogers) in any deployed CDS Invenio instance worldwide. Still, we have limited design discussions and testing to a core group of users:

- Present day catalogers at CERN.
- Future INSPIRE catalogers at the four laboratories CERN, DESY, Fermilab and SLAC.

### 6.2 User testing

#### 6.2.1 Design stage

In the early part of the design stage, catalogers at all involved laboratories were interviewed about their work processes. The results of these interview were gathered by participants in the INSPIRE collaboration and then documented on the project Twikis where they formed the foundation for the development of system requirements and design sketches. When this work was finished we took the results back to the catalogers and presented it to them. Comments and requests were collected and taken into account with further revisions being done.

The first INSPIRE workshop was also held during the design stage and was attended by project members familiar with the cataloger workflow, who could comment on recent developments in the design process.

#### 6.2.2 Development stage

During development every release was introduced to the cataloguer in all or some of these ways, depending on the significance of the release:

- Publishing on globally available development machines (CDSDEV / INSPIREDEV)
- Publishing on CERN production machine (CDSWEB)
- Local interactive presentation, demo and discussions at CERN and the other labs
- E-mail “newsletter”
- Twiki updates
- Talks with the catalogers
- Talks with INSPIRE project members

In semi-weekly phone- or videoconferences between members of the INSPIRE collaboration, the latest features were presented, discussed and feedback was given, often conveyed on behalf of catalogers at remote labs. Catalogers have also sent thorough feedback by e-mail after testing sessions of after encountering problems.

## 6.3 Handling of feedback and bugs

Feedback received during testing has mostly been requests for features that we either hadn't had time to implement or just hadn't thought of yet. In these cases the requests have been noted and possibly added to the feature list or, if already on the list, considered for an increased priority.

Sometimes a new feature doesn't perform according to the catalogers expectations. This is seldom critical and the issue is noted and corrected in the next release.

There have been quite a few bug reports along the way, most of them a result of actual bugs. Some critical bugs have been fixed and put in production the same day. Less dramatic bugs are listed and fixed before the next release.

## 6.4 Automated tests

Three different kinds of programmatic tests are being used in the CDS Invenio system [34]:

- **Unit tests**  
Tests important core functionality, should have no side-effects, can be written using Python unit testing framework.
- **Regressions tests**  
Tests overall application functionality. Regressions tests are allowed to have destructive side-effects. Can be written in regular Python or using the *mechanize* framework to write simple browser tests.
- **Web tests**  
Web tests are really a subset of regression tests with the characteristic being that they run scripted actions in a real browser. For CDS Invenio such tests are written and run using the Selenium IDE add-on for Firefox.

BibEdit is an administrative module and does not require the same level of testing as public modules. Still, having a wide range of tests is obviously profitable when developing, as it is easy to see if new developments have broken existing functionality. The problem is how to test a rather heavy Ajax application such as BibEdit.

It should be possible using Selenium web tests and I've also written a basic set of tests using this tool. These tests are testing the availability of the application and that the different levels of authorization works as intended. Naturally it is important that unauthorized users can't use the application to gain access to restricted material.

Nevertheless it would be desirable to write more tests, but here I ran into difficulties because of a bug in the version of Selenium IDE we're using, a product that unfortunately still is in Beta and haven't released a new version since June 2008 [35]. Basically the bug makes it impossible to get hold of the *window* object of the window where the test is being run and, since the window object is the global namespace for any JavaScript object, this makes it impossible to run tests on the JavaScript state of the application.

There are other frameworks for testing JavaScript and Ajax, but most seem fit to write unit tests rather than regression tests, which are what an advanced application like BibEdit needs the most.

## 6.5 Release routine

In an attempt to reduce the amount of mistakes being committed, to stay in line with Invenio development policies and to compensate some for the lack of good tests, I have tried to follow a routine of checks before any release of BibEdit consisting of the following steps:

- Browser performance and compatibility tests
  - IE7
  - IE8
  - FF2
  - FF3
  - Safari
  - Opera
  - Chrome
  - Iceape
- Pylint Python modules
- JSLint JavaScript files
- JSMIN JavaScript files
- Packaging – discuss new files, file names, directories with lead developer
- Copy new files onto development machines for user testing
- Commit to Git repository
- Copy new files onto production machines

Any problems encountered at any stage in this routine are naturally corrected and, if major changes had to be done, the routine restarted from scratch.



## 7. Project management

### 7.1 The project group

Contrary to most of my classmates my project was not completed as a group (when I say project group I'm thus referring to the whole Inspire project), but as a one-man job. Combining this with the large time scale, a full year, I think I've had a high degree of personal freedom in deciding how to best perform my work. And with freedom comes responsibility. If I were to do nothing for some weeks it's not entirely certain that anyone would have noticed, though in the long run the result of such laziness would of course be obvious.

However, I've always had the necessary guidance, communication or report lines readily available during my work. Usually my lead developer or other skilled colleagues has been accessible for any technical questions or problems I might have had. And for major decisions pertaining to the design or functionality of BibEdit, e-mail lists and Twiki-pages have been used intensively for discussing and reaching consensus, even with a project and user group spread around the world.

Additionally there have been weekly section meetings and semi-weekly project phone meetings or video conferences. In both settings I've had to report my progress and dealings in the time passed since the last meeting. We've also had two "Inspire-weeks" – workshops dedicated to analyzing, planning and discussing the progress of the Inspire project and several local meetings and demos with catalogers have been held both at CERN and the other laboratories (not by me, but with BibEdit as the topic of discussion).

### 7.2 Time management

I wouldn't say that there has been any aggressive time management going on in relation to my project. From the start I knew I had to deliver a minimum set of requirements sometime during the spring of 2009, and even that deadline was rather loose. Of course, if an unfinished module delays the whole project it certainly shouldn't be BibEdit, which is why I've from the start have had a continuous focus on finishing basic, critical functionality before anything else.

The project schedule shown in appendix A was developed as part of my pre-project report. With optimism not quite in touch with reality, I divided the remaining time into the number of time slots needed to fit all the features in the full module specification. I think my reasoning was that I could accomplish almost anything as long as I had strict deadlines to work towards.

It's now obvious to me that this is not how project schedules should be made. Even today, when the original specification has been split into four different components being worked on by three people and my project is nearing its end, we're still nowhere close to the goals set in this schedule. Of course, this is less catastrophic, as not all the features incorporated in this plan are required for Inspire to be production ready. Conclusions on the project plan and progress can be found in chapter 8.

### 7.3 Task management

Somewhere in the beginning of the project I started using the Twiki page of BibEdit to maintain an ever growing list of feature requests and pending tasks. I also had an *Ideas* section for more suggestions that would require more discussion before going on the feature list.

This had the advantage that everyone could see what I was working on at the moment and what I would be working on soon. If anyone got an idea for a new feature or found a bug, they could look on the Twiki to see if I was already aware of it and if not, alert me. And finally, the availability of this list made it easier to view and discuss the priority of the different jobs.

The advantage of this approach compared to any personal task managing list or system should be clear. Another option would be to use a full scale task and bug tracker like Savannah, but then the overhead of maintaining a complete list of, often small, features becomes too great and you need an additional system to keep track of all the little things. The Twiki list was simply a large table (like a digital blackboard) - easy to keep updated and with a low enough threshold for use that everything can be registered there. At least it worked nicely for a rather small project like mine.

## 8. Conclusion

I've spent approximately 8 months on this bachelor project with the objective of creating a heavily upgraded version of the CDS Invenio record editor, BibEdit, fit for use in the INSPIRE cataloging workflow.

The old BibEdit module has been completely rebuilt. In the backend some design concepts have been kept as they were, but the entire codebase has been more or less redone. The result is now a BibEdit application which is much superior to the old version when it comes to offering a smooth and professional cataloging interface. By using a modern Ajax supported web interface it allows for easy and efficient editing of a record. It is also built in such a way that adding additional features, which there no doubt will be a demand for, should be a relatively simple task

All the basic and some advanced features concerning the core and editing functionality are now in place and BibEdit has been integrated in the production environment at CERN and in the CDS Invenio official source code repository. However BibEdit is not a finished product and it should be noted that a few, but significant requirements have not been implemented in the duration of my project. I'm thinking in particular of the integration with the other modules in the cataloging workflow, like BibCatalogue, BibCheck and BibKnowledge. These modules have also been in development and their interfaces were finished only weeks ago, still I've been so busy implementing other high priority features that I haven't really missed them. This has the ramification that BibEdit is not yet ready for use in INSPIRE. However at the time of writing it is only still April and I will be working dedicated on finishing these features until the end of June, so my hope is that they will be ready well before INSPIRE is going into production use.

If we compare the progress of the project with the original project schedule *a lot* has changed. The delegation of large parts of the project, like *MultiEdit*, to other developers is explained in detail in chapter 2.1. The original project schedule assumed 20 day-release cycles for each prototype version, encompassing coding, testing and integration, for a total of three prototypes before Christmas. Even in a perfect world with no distractions or unexpected delays, that is too optimistic when each prototype contains many large, new features. Also not taken into account was the flood of new, high priority feature requests I've been receiving ever since the user testing of the first prototype.

This means approximately a two month delay compared with the original schedule, and the integration with the rest of the cataloging modules has been the victim of this delay. The *diff-and-merge* functionality cannot be created anyway until someone else does a major piece of work on the BibUpload module (see chapter 4.2.3).

If we look at the risk analysis we're still in rather good shape. Risks with both high impact and high probability have been avoided. The project doesn't seem to completely meet the original deadline, but the INSPIRE deadlines seem in general to be drifting a few months ahead anyway. The majority of the important requirements are in place and work will soon commence on those that are not.

It also looks like the concerns about the Ajax based solution being either too slow, too heavy or having too poor usability can be relaxed. As planned, an early working prototype was created so these risks could be clarified and I did user testing of early design and prototypes to avoid any surprises later on. Of course I've been lucky too; the Ajax technology has performed perhaps better than expected and the project has coincided with dramatic improvements in the speed of JavaScript interpreters and fierce competition in the browser market – both things have played on our side.

Modules that BibEdit uses or depends on have been delayed, but so has BibEdit itself, eliminating the last two risks on the list in a less gracious way.

The conclusion should mention any problems that have influenced the project, but few things feel significant enough to mention. Continuously changing requirements - the ghost that haunts all complex software projects - has been present here too. Although I don't think it has been that detrimental to my work. Most of the changes are new ideas and feature requests from users being inspired by playing with the prototypes and is clearly very useful feedback. Also since the design process hasn't been too rigid it's not that difficult to incorporate these suggestions into the product.

Apropos the design process; I've fallen victim to a few faulty assumptions during the design talks. The consequences of these misunderstandings have taken me some time to set right, and I wonder if they could have been avoided. Although I doubt a more extensive design process would be worth the price for a project this size.

BibEdit as it functions today shows that it's possible to make a decent web application by using Ajax technology, even with limited time and resources. INSPIRE will enable catalogers to cooperate on a global level using high-availability web tools, performing close to what they could expect from desktop applications, to provide high-energy physicists with quality data in their field of research.

## 9. Literature

### 9.1 References

- [1] Wikipedia. *Particle Physics*. Available from: [http://en.wikipedia.org/wiki/Particle\\_physics](http://en.wikipedia.org/wiki/Particle_physics) (30.03.2009)
- [2] CERN (2008) *CERN in a nutshell*. Available from: <http://public.web.cern.ch/public/en/about/About-en.html> (30.03.2009)
- [3] Wikipedia. *CERN*. Available from: <http://en.wikipedia.org/wiki/CERN> (30.03.2009)
- [4] CERN (09.02.2009) *Press release: CERN management confirms new LHC restart schedule*. Available from: <http://press.web.cern.ch/press/PressReleases/Releases2009/PR02.09E.html> (30.03.2009)
- [5] CERN Document Server Software Consortium (08.08.2005) *CDS Invenio Overview*. Available from: <http://cdsware.cern.ch/invenio/index.html> (30.03.2009)
- [6] CERN Document Server Software Consortium (08.08.2005) *CDS Invenio Demo*. Available from: <http://cdsware.cern.ch/invenio/demo.html> (30.03.2009)
- [7] Gentil-Beccot, A. et al. *Information Resources in High-Energy Physics : Surveying the Present Landscape and Charting the Future Course*. J. Am. Soc. Inform. Sci. Technol. 60, 1 (2009) pp.150–160. Available from: <http://cdsweb.cern.ch/record/1099955> (31.03.2009)
- [8] Wikipedia. *Stanford Physics Information Retrieval System*. Available from: <http://en.wikipedia.org/wiki/SPIRES> (31.03.2009)
- [9] Holtkamp A. (2007) *Inspire Overview* (slides). HEP Information Resource Summit (20-22 May 2008). Available from: <https://indico.desy.de/materialDisplay.py?contribId=6&sessionId=18&materialId=slides&confId=800> (31.03.2009)
- [10] DESY (Hamburg, Germany - 2008) *High-Energy Physics Labs Join to Build a New Scientific Information System* (press release). Available from: <https://indico.desy.de/getFile.py/access?resId=1&materialId=3&confId=800> (31.03.2009)
- [11] CERN Document Server Software Consortium (08.08.2005) *CDS Invenio Features*. Available from: <http://cdsware.cern.ch/invenio/features.html> (31.03.2009)
- [12] Wikipedia. *Academic publishing*. Available from: [http://en.wikipedia.org/wiki/Academic\\_publishing](http://en.wikipedia.org/wiki/Academic_publishing) (31.03.2009)
- [13] SCOAP3. Front page. Available from: <http://scoap3.org/> (31.03.2009)
- [14] Cazenovia College. *Glossary of Library Terms*. Available from: <http://www.cazenovia.edu/Default.aspx?tabid=703> (03.04.2009)
- [15] Wikipedia. *Lamp (software bundle)*. Available from: [http://en.wikipedia.org/wiki/LAMP\\_\(software\\_bundle\)](http://en.wikipedia.org/wiki/LAMP_(software_bundle)) (06.04.2009)

- [16] CDS Invenio Twiki. *Python Advocacy*. Available from: <https://twiki.cern.ch/twiki/bin/view/CDS/PythonAdvocacy> (06.04.2009)
- [17] Wikipedia. *Git (software)*. Available from: [http://en.wikipedia.org/wiki/Git\\_\(software\)](http://en.wikipedia.org/wiki/Git_(software)) (06.04.2009)
- [18] CDS Invenio Twiki. *Git Advocacy*. Available from: <https://twiki.cern.ch/twiki/bin/view/CDS/GitAdvocacy> (06.04.2009)
- [19] Oliver Steele. *My Git Workflow*. Available from: <http://osteele.com/archives/2008/05/my-git-workflow> (06.04.2009)
- [20] Wikipedia. *JSON*. Available from: <http://en.wikipedia.org/wiki/JSON> (06.04.2009)
- [21] CDS Invenio Twiki. *Emacs Advocacy*. Available from: <https://twiki.cern.ch/twiki/bin/view/CDS/EmacsAdvocacy> (06.04.2009)
- [22] Google Code. *js2-mode*. Available from: <http://code.google.com/p/js2-mode/> (06.04.2009)
- [23] Ajax Patterns. *Unique URLs*. Available from: [http://ajaxpatterns.org/wiki/index.php?title=Unique\\_URLs](http://ajaxpatterns.org/wiki/index.php?title=Unique_URLs) (10.04.2009)
- [24] mozdev.org. *mozex*. Available from: <http://mozex.mozdev.org/index.html> (10.04.2009)
- [25] Humanized. Weblog: *Undo Made Easy with Ajax (Part 1)*. Available from: <http://humanized.com/weblog/2007/09/14/undo-made-easy-with-ajax-part-1/> (10.04.2009)
- [26] Humanized. Weblog: *Undo Made Easy with Ajax (Part 2): Time-Sensitive Actions*. Available from: [http://humanized.com/weblog/2007/10/22/undo\\_with\\_ajax\\_2/](http://humanized.com/weblog/2007/10/22/undo_with_ajax_2/) (10.04.2009)
- [27] W3schools. Differences Between XHTML And HTML. Available from: [http://www.w3schools.com/XHTML/xhtml\\_html.asp](http://www.w3schools.com/XHTML/xhtml_html.asp) (10.04.2009)
- [28] json.org. *JSON in JavaScript*. Available from: <http://www.json.org/js.html> (13.04.2009)
- [29] Jeditable. *Edit In Place Plugin For jQuery*. Available from: <http://www.appelsiini.net/projects/jeditable> (12.04.2009)
- [30] jQuery Plugins. *Auto Growing Textareas*. Available from: <http://plugins.jquery.com/project/autogrow> (12.04.2009)
- [31] JavaScriptly. *Quick & Useful JQuery Plugins*. Available from: <http://javascriptly.com/2008/09/quick-useful-jquery-plugins/> (12.04.2009)
- [32] QuirksMode.org. *Detecting keystrokes*. Available from: <http://www.quirksmode.org/js/keys.html> (13.04.2009)
- [33] Google Code. *Javascript jQuery Hotkeys Plugin*. Available from: <http://code.google.com/p/js-hotkeys/> (12.04.2009)

[34] CDS Invenio Hacking Guide. *Test Suite Strategy*. Available from:  
<http://cdsweb.cern.ch/help/hacking/test-suite> (06.04.2009)

[35] SeleniumHQ. *Downloads*. Available from: <http://seleniumhq.org/download/> (06.04.2009)

## 9.2 Web sites of interest

### 9.2.1 Project and background material

- **Project blog:** <http://raae-bachelor.blogspot.com/>
- **CERN:** <http://www.cern.ch>  
The web site of CERN.
- **CDS Invenio:** <http://cdsware.cern.ch/invenio/index.html>  
The web site of the CDS Invenio platform.
- **CDS Invenio Demo:** <http://invenio-demo.cern.ch/>  
The CDS Invenio demo site.
- **CERN Document Server:** <http://cdsweb.cern.ch>  
The CERN Document Server, running on the CDS Invenio platform.
- **CDS Invenio Savannah:** <https://savannah.cern.ch/projects/cdsware/>  
Savannah task tracker for CDS Invenio.
- **CDS Invenio Twiki:** <https://twiki.cern.ch/twiki/bin/view/CDS/Invenio>  
Twiki collaboration wiki for CDS Invenio.
- **MARC 21 Format for Bibliographic data:**  
<http://www.loc.gov/marc/bibliographic/ecbdlist.html>  
MARC 21 field list.
- **MARC21 XML Schema:** <http://www.loc.gov/standards/marcxml/>
- **Inspire:** <http://www.hep-inspire.net/>  
Inspire alpha (showcase) site.
- **Inspire Savannah:** <https://savannah.cern.ch/projects/inspire/>  
Savannah task tracker for Inspire.
- **Inspire Twiki:** <https://twiki.cern.ch/twiki/bin/view/Inspire/WebHome>  
Twiki collaboration wiki for Inspire.
- **BibEdit Twiki page:** <https://twiki.cern.ch/twiki/bin/view/Inspire/SystemDesignBibEdit>  
Twiki page for BibEdit (in an Inspire context).
- **Spires:** <http://www-spires.fnal.gov/>  
Spires high energy physics literature database.
- **arXiv.org:** <http://arxiv.org/>  
arXiv.org e-print archive



### 9.2.2 Technical resources

- **Core JavaScript 1.5 Guide:** [https://developer.mozilla.org/en/Core\\_JavaScript\\_1.5\\_Guide](https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide)
- **Core JavaScript 1.5 Reference:**  
[https://developer.mozilla.org/en/Core\\_JavaScript\\_1.5\\_Reference](https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference)
- **Ajax Patterns:** <http://ajaxpatterns.org/>
- **jQuery:** <http://jquery.com/>  
The jQuery JavaScript library.
- **QuirksMode:** <http://www.quirksmode.org/>  
«The prime source for browser compatibility information on the Internet. »  
«Hell is other browsers. »

### 9.3 Books

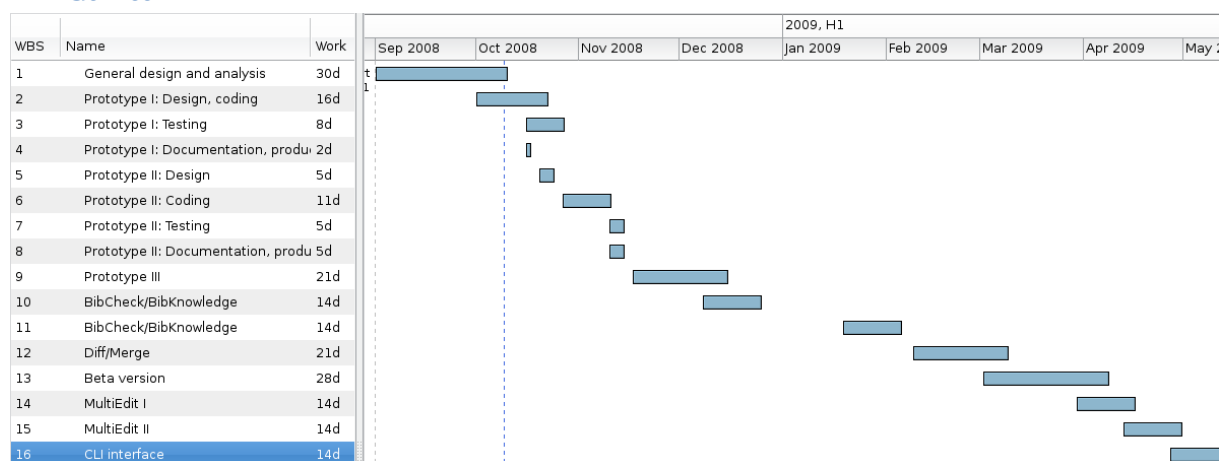
- Sobell, M. G. (2005) A Practical Guide to Linux Commands, Editors, and Shell Programming. Prentice Hall PTR.
- Lutz, M. & Ascher, D. (2003) Learning Python. O'Reilly Media, Inc.
- Negrino, T. (2007) JavaScript and Ajax For the Web. Peachpit Press.
- Schmitt, C. (2006) CSS Cookbook. O'Reilly Media, Inc.
- Flanagan, D. (2002) JavaScript: the definitive guide. O'Reilly Media, Inc.

## Appendix A: Project schedule

### A.1 Deliverables

Name	Description
Prototype I	Today's BibEdit with JS-supported UI and client-side HTML generation and dynamic processing.
Prototype II	Field validation, internationalization, hotkeys, dialogs, 'Save as new', prettifying.
Prototype III	User settings dialog, filters and compressed views, performance improvements.
BibCheck / BibKnowledge	Interfaces and display of BibCheck/BibKnowledge data, including autosuggest and related UI upgrades.
Diff/Merge	Committing of change sets, handling of the BibEdit 'on submit merging'.
Beta version	Logging, better AJAX control (concurrent requests, error handling, integrity checking, handling of users without adequate script or browser support), interface with cataloger interface (if ready).
MultiEdit I	Automation of the search, wget, search and replace text and upload approach.
MultiEdit II	A more refined interface and a more rational approach to multi-editing.
CLI interface	Update as necessary, with new (MultiEdit) or removed (history support?) features.

### A.2 Gantt



## Appendix B: Risk Assessment

Levels: Low, medium or high.

#	Risk	Probability	Impact	Overall impact
1	Project not meeting requirements on deadline.	Medium	High	Medium
2	AJAX based solution too slow / too heavy.	High	High	High
3	Usability of final solution unsatisfactory for users.	High	High	High
4	Some module which BibEdit will use not finished on time.	High	Low	Medium
5	Some module which BibEdit depends on not finished on time.	Medium	High	Medium

As one can see all these risks have a significant overall impact (a result of probability and impact combined). Here is how they will be addressed:

**Risk #1:** The requirements can be split into three groups: Those that are absolutely necessary for Inspire adoption, those that are important, but not needed to take the system into production and those that would be nice to have. This is also an ascending ordering of the groups according to size. If the project is delayed one will have to delay implementation of requirements from the last, and in worst case, from the second last group.

**Risk #2 and #3:** #2 is a very serious risk caused by the experimental nature of the project, at least compared with most other modules which is written in serverside Python, a proven concept. #3 can be caused by #2, but also by other factors like bad design, buggy code or simply by making the “wrong” product. Both risks will be addressed by iterative system development and frequent prototyping combined with user testing and feedback.

**Risk #4:** BibEdit uses several modules and there is a risk that one or more of them might not be finished on time. BibEdit should therefore be written in a way that let's these modules be plugged/unplugged easily, making BibEdit independent of their presence.

**Risk #5:** BibEdit has no direct dependencies, but it's main use case (the cataloger workflow) depends on the completion of the cataloger interface. That is outside my control, but BibEdit can still be used as a standalone tool should this critical module not be finished in time.

## Appendix C: User Documentation

### BibEdit Keyboard Shortcuts

#### Basic record actions

Shortcut	Definition	Action
g	Select	Go to the record selection field
Shift+s	Submit	Submit the record.
Shift+c	Cancel	Cancel editing of the record.
Shift+d	Delete	Delete the record.
Shift+t	Tags	Toggle MARC/human tags.
a	Add field	Add a new, empty field.
o	Sort fields	Sort fields. This simply updates the view, it doesn't change the record itself.
Del	Delete selected	Deletes all selected fields.

### Focused (clicked) subfield

Shortcut	Definition	Action
Tab	Next	Move focus to next subfield.
Shift+Tab	Previous	Move focus to previous subfield.
Return	Edit	Edit focused subfield.
Space	Select	Select focused subfield.
Shift+Space	Select field	Select parent field of focused subfield.
Ctrl+Up	Move up	Move focused subfield up.
Ctrl+Down	Move down	Move focused subfield down.
Del	Delete	Delete focused subfield.
Shift+Del	Delete field	Delete parent field of focused subfield.
Ctrl+Shift+e	Add subfield	Add an additional subfield at the end.
Ctrl+Shift+d	Remove subfield	Remove a subfield from the end.

### Input field/form

Shortcut	Definition	Action
Ctrl+Shift+s	Save	Save content of field/form.
Ctrl+Shift+s	Cancel	Cancel editing of field/form.
Ctrl+Shift+s	Clear	Clear content of field/form.
Ctrl+Shift+e	Add subfield	Add an additional subfield at the end.
Ctrl+Shift+d	Remove subfield	Remove a subfield from the end.

### Other functionality

Shortcut	Definition	Action
s	Selection mode	Toggle selection mode.