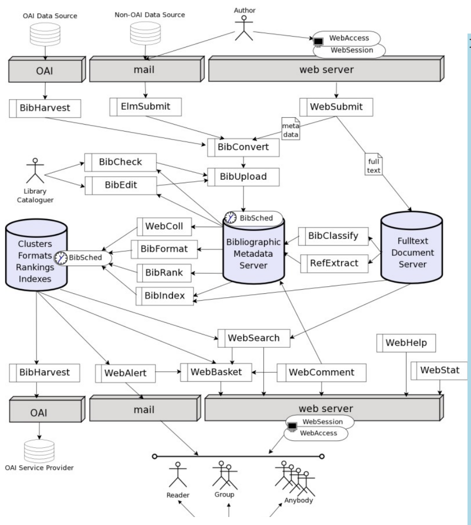
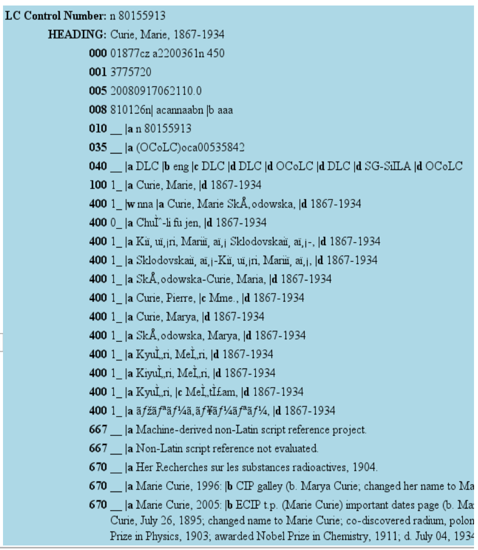
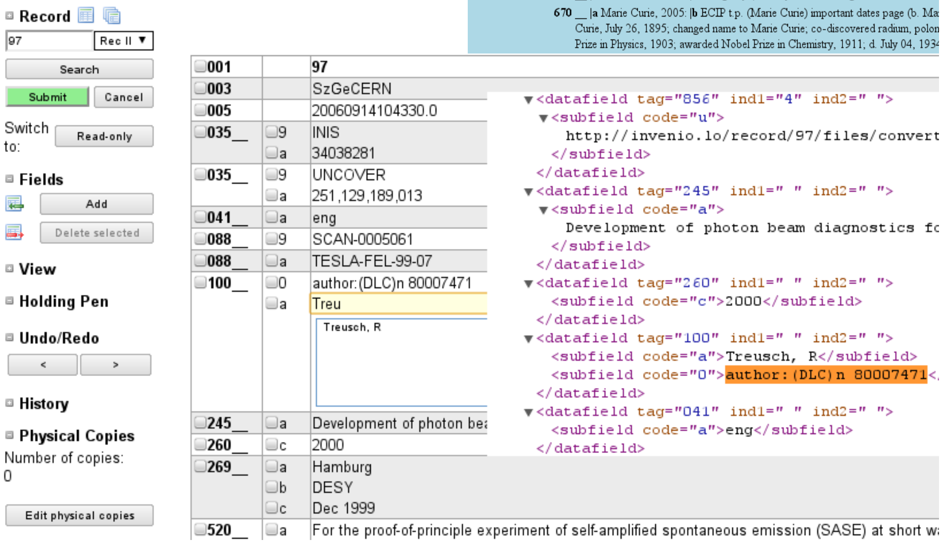
Authority Control for INVENIO





LC Control Number: n 80155913
HEADING: Curie, Marie, 1867-1934
000 01877cz a2200361n 450
001 3775720
005 20080917062110.0
008 810126n| acannaabn |b aaa
010 __ |**a** n 80155913
035 __ |**a** (OCoLC)oca00535842
040 __ |**a** DLC |**b** eng |**c** DLC |**d** DLC |**d** OCoLC |**d** DLC |**d** SG-SiILA |**d** OCoLC
100 1_ |**a** Curie, Marie, |**d** 1867-1934
400 1_ |**w** nna |**a** Curie, Marie SkÅ‚odowska, |**d** 1867-1934
400 0_ |**a** ChuÌˆ-li fu jen, |**d** 1867-1934
400 1_ |**a** Kiï, uï,¡ri, Mariiï, aï,¡ Sklodovskaiï, aï,¡-, |**d** 1867-1934
400 1_ |**a** Sklodovskaiï, aï,¡-Kiï, uï,¡ri, Mariiï, aï,¡, |**d** 1867-1934
400 1_ |**a** SkÅ‚odowska-Curie, Maria, |**d** 1867-1934
400 1_ |**a** Curie, Pierre, |**c** Mme., |**d** 1867-1934
400 1_ |**a** Curie, Marya, |**d** 1867-1934
400 1_ |**a** SkÅ‚odowska, Marya, |**d** 1867-1934
400 1_ |**a** KyuÌ„ri, MeÌ„ri, |**d** 1867-1934
400 1_ |**a** KiyuÌ„ri, MeÌ„ri, |**d** 1867-1934
400 1_ |**a** KyuÌ„ri, |**c** MeÌ„tÌ£am, |**d** 1867-1934
400 1_ |**a** ãƒžãƒªãƒ¼ã‚ãƒ¥ãƒªãƒ¼, |**d** 1867-1934
667 __ |**a** Machine-derived non-Latin script reference project.
667 __ |**a** Non-Latin script reference not evaluated.
670 __ |**a** Her Recherches sur les substances radioactives, 1904.
670 __ |**a** Marie Curie, 1996: |**b** CIP galley (b. Marya Curie; changed her name to Ma
670 __ |**a** Marie Curie, 2005: |**b** ECIP t.p. (Marie Curie) important dates page (b. Ma
Curie, July 26, 1895; changed name to Marie Curie; co-discovered radium, polon
Prize in Physics, 1903; awarded Nobel Prize in Chemistry, 1911; d. July 04, 1934



001		97
003		SzGeCERN
005		20060914104330.0
035__	9	INIS
	a	34038281
035__	9	UNCOVER
	a	251,129,189,013
041__	a	eng
088__	9	SCAN-0005061
088__	a	TESLA-FEL-99-07
100__	0	author:(DLC)n 80007471
	a	Treu Treusch, R
245__	a	Development of photon bea
260__	c	2000
269__	a	Hamburg
	b	DESY
	c	Dec 1999
520__	a	For the proof-of-principle experiment of self-amplified spontaneous emission (SASE) at short wa

```
▼<datafield tag="856" ind1="4" ind2=" ">
  ▼<subfield code="u">
     http://invenio.lo/record/97/files/convert
   </subfield>
 </datafield>
▼<datafield tag="245" ind1=" " ind2=" ">
  ▼<subfield code="a">
     Development of photon beam diagnostics fo
   </subfield>
 </datafield>
▼<datafield tag="260" ind1=" " ind2=" ">
   <subfield code="c">2000</subfield>
 </datafield>
▼<datafield tag="100" ind1=" " ind2=" ">
   <subfield code="a">Treusch, R</subfield>
   <subfield code="0">author:(DLC)n 80007471<
 </datafield>
▼<datafield tag="041" ind1=" " ind2=" ">
   <subfield code="a">eng</subfield>
 </datafield>
```

Bachelor Project 2011
Christopher Dickinson (student)
Omar Abou Khaled (prof.)
Elena Mugellini (prof.)
Pascal Felber (expert)

Table of Contents

1 Introduction

1.1 General

This is the final report for a Bachelor project at the École d'Ingénieurs et d'Architectes, in collaboration with the European Organization for Nuclear Research (CERN) near Geneva. The official starting date for this project was June 30th 2011. All of the work described in this document was done at CERN during an internship in the IT-UDS-CDS team.

1.2 Invenio

INVENIO[1] is a free, open-source software for running digital libraries or document repositories on the web. It was originally developed at CERN to run the CERN document server, where it currently manages over 1'000'000 bibliographic records in high-energy physics since 2002, covering articles, books, journals, photos, videos, and more. Currently, INVENIO is being co-developed by an international collaboration comprising institutes such as CERN, DESY, EPFL, FNAL, SLAC and is being used by about thirty scientific institutions worldwide[2]. As underlying bibliographic format, it uses the so called MARC 21[3] standard which is still the main international standard for large-scale digital library management systems.

1.3 The Goal of the Project: Authority Control

The goal of the project was to add Authority Control[4] to INVENIO. Authority control provides a library management software with two main functions. 1. It allows the disambiguation between similar or identical terms, such as author names referring to different people. 2. It allows for the collocation of seemingly distinct information that logically belongs together, e.g. alternate names for an author or institution. In recent years, INVENIO users have seen the need to control standardized ways of managing the names of authors and institutions, journals and subjects. “Authority records” keep track of the standard way (e.g. “Curie, Marie”) as well as alternative ways of writing a name (e.g. “Skłodowska Curie, Marie” or “Skłodowska, Marya”) and also help disambiguate between multiple authors/institutions etc with the same name. These records can be used by librarians as a reference for those fields in “bibliographic records” that require such a standardization.

1.4 The INVENIO Team(s)

The internship associated with this Bachelor project was done as part of the IT-UDS-CDS team on the Meyrin site of CERN. Jean-Yves Le Meur (F) is the head of this team and project manager for INVENIO. Tibor Simko (CZ) is the head developer and main architect of INVENIO. Jerome Caffaro (CH) and Samuele Kaplun (IT) are among the more experienced developers for INVENIO. There are also a number of developers working in the same team for a limited duration of between 3 and 12 months.

1 http://invenio-software.org/
2 http://invenio-software.org/wiki/General/Demo
3 http://en.wikipedia.org/wiki/MARC_standards#MARC_21
4 http://en.wikipedia.org/wiki/Authority_control

Besides the team just mentioned, there are various other teams involved in the development of INVENIO. Various developers contributing to INVENIO work for the GS-SIS group at the Meyrin Site of CERN, responsible for managing the Library as well as the Historical and Scientific Archives of CERN. Developers working for the INSPIRE[5] project also contribute back to INVENIO, since INSPIRE is built upon the INVENIO software.

Finally, there are organizations that use INVENIO for their own digital libraries (e.g. the FZ Jülich[6]), and that besides the occasional contribution to the code base are in close communication with the main INVENIO team and contribute with their feature suggestions which often find their way into new releases of the software.

All of the teams mentioned played a role in the planning and coordination of the new Authority Control module and had an interest in it's completion and use in production.

1.5 The Structure of this Document

This document presents the work done on this project according to the main development phases Analysis, Design, Implementation and Tests. The Conclusion section looks back at the results and experiences learned during the project as well as offering an outlook of possible future improvements to be made to INVENIO's authority control module. An illustration index is given for an overview and references of the illustrations used throughout the document. The glossary explains the most current terms and abbreviations related to INVENIO. The contents of the CD-ROM are included in a separate sections. And finally, the Appendices Section contains the various appendices to this document.

Besides being a technical report, it also includes some description of the experiences made at CERN. The latter should help in evaluating not only the technical work accomplished, but also give some insight into the project management and soft skills required.

The report was created using OpenOffice 3.2. The decision to use this rather than the much more modern Microsoft Word was based upon the fact that everyone at IT-UDS-CDS uses some form of Linux and hence would prefer a document format which is truly platform independent. Compared to a purely academic project, this document contains only very few diagrams or images. This is due mainly to the way work is done in the INVENIO teams (very few diagrams, many iterative prototype cycles).

5 http://www.projecthepinspire.net/
6 http://www.fz-juelich.de/

2 Preparations

This section describes the various preparations that were necessary at the very beginning of the project and that preceded the Analysis phase.

2.1 Arrival at CERN

Upon arrival at CERN, I learned that all of the administrative paperwork that is usually done a few weeks before an intern arrives, had in my case been forgotten. This meant I had no CERN card (and hence no authorization to enter the Meyrin site of CERN unaccompanied – and no authorization to enter through the much closer French gate at all, no possibility of renting a bike in time before the arrival of the summer students, no authorization to rent books from the library), no computing account (and hence no access to the INVENIO developer mailing list and chat rooms, no space on the lxplus grid to host my private and public GIT repository and code backups, no account to create tickets and merge requests on the team's TRAC system[7], no access to the virtual windows machines for printing, etc.) and no key to our shared office. Until this was finally done 7 days after my arrival, I spent a great deal of time trying to do as much as I possibly could to make sure this work was done as quickly as possible and trying to find alternatives and workarounds in the mean time so that I could continue in my work.

2.2 Scope statement

During the first 2 weeks of the internship, there was no scope statement either. Finding a date with the project manager and the head developer in order to discuss the detailed project requirements was one of the first non-administrative tasks that fell into the responsibility of this internship, a task which was not easily accomplished due to the frequent absences and busy schedule of the parties involved. Since none of the requirements could be obtained in written form from the project management, writing the scope statement was also one of the early tasks of this project.

2.3 Making the most out of it

In order to make the best possible use of the time until the project description meeting could be arranged, various activities were completed in this period:

First of all, the latest version of the INVENIO software had to be pulled from the central git repository and installed locally on a Debian machine. Thanks to previous experience with Debian Linux, GIT, Apache 2 and MySQL, this was accomplished fairly easily with the help of on-line installation instructions, and in the case of conflicting third-party program dependencies with the direct help of Tibor Simko. [8]

Secondly, since the INVENIO in its current form is written mainly in Python, part of the preparation consisted in becoming familiar with this language and the development work-flow associated with it. One of the extremely useful tutorials for Python as such was found in Google's Python Class[9] which offers tutorials combined with coding exercises and solutions that can be downloaded and

7 http://invenio-software.org/query
8 For future reference, the most up to date installation instructions are not to be found on-line, but in the README file that ships with the software bundle.
9 http://code.google.com/edu/languages/google-python-class/

run locally.

Both of these requirements were completed before the actual beginning of the internship for the most part, but the extended delays before receiving the detailed project description allowed for a more in-depth study in both areas.

Finally, the first 2 weeks were also used to become familiar with the coding style and work-flow customs of the various INVENIO developer teams as described in numerous public web pages dedicated to this subject[10]. This was useful and necessary in order to adopt the proper coding style and work-flow from the outset and not to run into problems with the way the team was used to working later on.

Throughout this period, various members of the IT-UDS-CDS team were very helpful and friendly in regard to questions about INVENIO, Python, command-line GIT and related subjects. Towards the end of the preparation time, Jerome Caffaro offered to give me a 1h introduction to INVENIO based on the on-line tutorials mentioned above.

10 Most of them can be found by starting at https://twiki.cern.ch/twiki/bin/view/CDS/Invenio.

3 Analysis

3.1 Data collection

Since the writing of the scope statement was part of this project, much of the Analysis phase was done as part of the process of inquiring about the detailed project requirements before the scope statement was written. In addition, however, various meetings were held with key users of INVENIO in order to understand their unique needs for this new Authority module. One of these people was Jocelyne Jerdelet from the CERN library who used to work with a commercial library management software called ALEPH and who now uses INVENIO in stead. She was able to explain the work-flow she was accustomed to from ALEPH for the use and management of authority records, but also of the limitations and frustrations encountered with the previous system. This gave me a good idea of what kind of the kind of user interface that was expected as well as of the improvements that would be necessary compared to the older system.

Another meeting was held with Alexander Wagner from the Forschungszentrum Jülich[11], Germany, who happened to be in Geneva for a convention and who had the time to see me for a meeting in the city. He was one of the main inspirations for adding authority control to INVENIO since his institution has a strong need for it, esp. related to their various institutions who want to know across institutional hierarchies and historical mergers and divisions which publications can be attributed to which institutions. Thanks to this talk I was able to get a very precise idea of the requirements and use cases to expect.

3.2 A brief introduction to MARC 21

In order to understand the basics of what is to follow, it is important to have at least a rudimentary understanding of some of the main properties of the MARC notation.

MARC records are composed of fields, indicators and subfields. We will use a demo bibliographic record from the default INVENIO installation to demonstrate these 3 types.

Demo bibliographic record in MARC 21 format:

```
001__ 97
003__ SzGeCERN
005__ 20060914104330.0
035__ $$9INIS$$a34038281
035__ $$9UNCOVER$$a251,129,189,013
041__ $$aeng
088__ $$9SCAN-0005061
088__ $$aTESLA-FEL-99-07
100__ $$aTreusch, R
245__ $$aDevelopment of photon beam diagnostics for VUV radiation from a
SASE FEL
269__ $$aHamburg$$bDESY$$cDec 1999
260__ $$c2000
595__ $$aSIS INIS2004
595__ $$aSIS UNC2002
595__ $$dDevelopment of photon beam diagnostics for VUV radiation from a
SASE FEL
```

11 http://www.fz-juelich.de/

```
65017 $$2SzGeCERN$$aAccelerators and Storage Rings
690C_ $$aARTICLE
694__ $$9INIS$$aParticle accelerators
695__ $$9INIS$$aphoton-beams
700__ $$aLokajczyk, T
700__ $$aXu, W
700__ $$aJastrow, U
700__ $$aHahn, U
700__ $$aBittner, L
700__ $$aFeldhaus, J
773__ $$c456-462$$n1-3$$pNucl. Instrum. Methods Phys. Res., A$$v445$$y2000
8564_ $$uhttp://invenio.lo/record/97/files/convert_SCAN-0005061.pdf
916__ $$sn$$w200430
960__ $$a13
961__ $$c20061230$$h0016$$lCER01$$x20040727
962__ $$b000289917$$k456-462$$nhamburg990823
963__ $$aPUBLIC
970__ $$a002471378CER
980__ $$aARTICLE
```

The example shown above is taken from the default INVENIO installation and represents the meta-data for an article with the title "Development of photon beam diagnostics for VUV radiation from a SASE FEL". As one can see, this title is contained within the MARC data on the line that begins with **245__ $$a**. **245** is a so called **field**, whereas **$$a** is called a **subfield**. The indicators are the two digits that follow the field, in this example they are mostly empty, which looks like "__". In INVENIO the indicator fields are mostly left empty. All values are contained within the subfields. Here the author names have been highlighted. The main author name is contained in the **100__a** subfield, the other author names are contained within the **700__a** subfields. The full list of MARC 21 field/subfield to human readable field names mapping for bibliographic records can be found at http://www.loc.gov/marc/bibliographic/ecbdlist.html (the same lists exist for authority records as well).

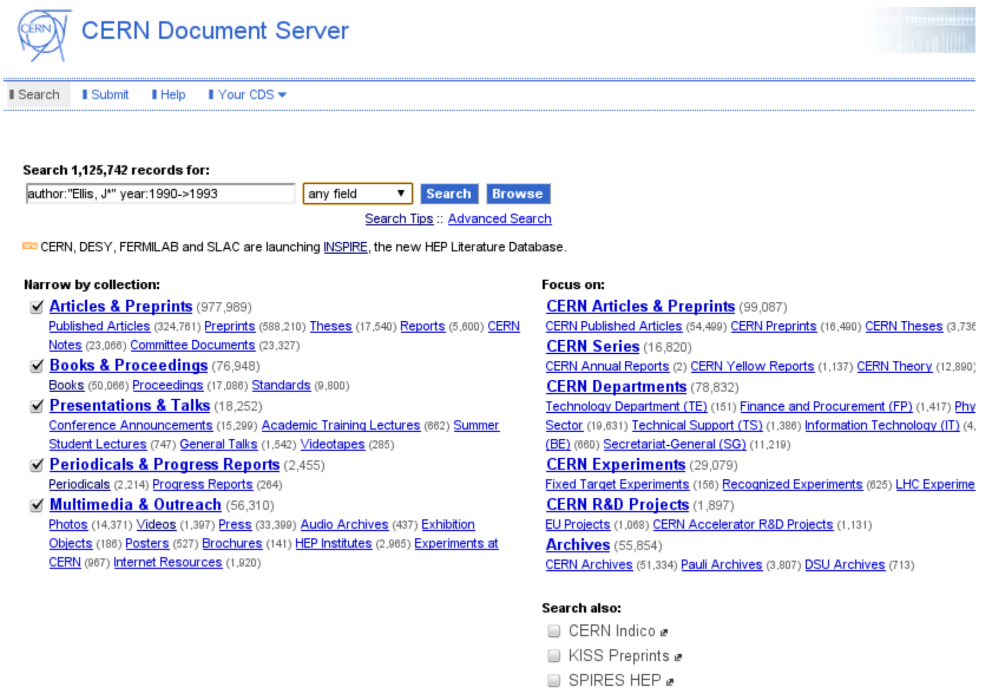
Although authority records have the same basic MARC format, it is important to know that identical field numbers may have different meanings in authority records vs. bibliographic records. E.g. field 500 is used for 'General Note' in bibliographic records, whereas in authority records it is used for a personal name 'see also from' information, typically referring to separate author name authority records.

Note: The number code of a field, e.g. '245' is often referred to as a 'tag'. In INVENIO, the whole field-subfield combination is often simply referred to as a 'field' or 'tag' and is written in this format: 'XXX__x'.

3.3 Collections in INVENIO

Note 2: Since INVENIO has no way to know if a particular MARC record is a bibliographic or authority record, it will treat authority records as bibliographic records. This introduces some problems, which need to be addressed by this project. In order for INVENIO to have an easy way to know if a record is an authority record or not, we will introduce a value 'AUTHORITY' into the MARC subfield '980__a', the subfield used by INVENIO to designate which collection a MARC records belongs to. An additional '980__a' subfield will contain the type of authority record (e.g. 'AUTHOR'). For more details on this notation, cf. the user manuals for librarians and for INVENIO administrators.

Illustration 1 below shows INVENIO collections as they appear on the CDS Document Server



Search 1,125,742 records for:

author:"Ellis, J*" year:1990->1993

Search Tips :: Advanced Search

CERN, DESY, FERMILAB and SLAC are launching INSPIRE, the new HEP Literature Database.

Narrow by collection:

✓ **Articles & Preprints** (977,989)
Published Articles (324,761) Preprints (588,210) Theses (17,540) Reports (5,600) CERN Notes (23,066) Committee Documents (23,327)

✓ **Books & Proceedings** (76,948)
Books (50,066) Proceedings (17,086) Standards (9,800)

✓ **Presentations & Talks** (18,252)
Conference Announcements (15,299) Academic Training Lectures (662) Summer Student Lectures (747) General Talks (1,542) Videotapes (285)

✓ **Periodicals & Progress Reports** (2,455)
Periodicals (2,214) Progress Reports (264)

✓ **Multimedia & Outreach** (56,310)
Photos (14,371) Videos (1,397) Press (33,399) Audio Archives (437) Exhibition Objects (186) Posters (527) Brochures (141) HEP Institutes (2,965) Experiments at CERN (967) Internet Resources (1,920)

Focus on:

CERN Articles & Preprints (99,087)
CERN Published Articles (54,499) CERN Preprints (16,490) CERN Theses (3,73

CERN Series (16,820)
CERN Annual Reports (2) CERN Yellow Reports (1,137) CERN Theory (12,890

CERN Departments (78,832)
Technology Department (TE) (151) Finance and Procurement (FP) (1,417) Phy
Sector (19,631) Technical Support (TS) (1,386) Information Technology (IT) (4,
(BE) (660) Secretariat-General (SG) (11,219)

CERN Experiments (29,079)
Fixed Target Experiments (156) Recognized Experiments (625) LHC Experime

CERN R&D Projects (1,897)
EU Projects (1,068) CERN Accelerator R&D Projects (1,131)

Archives (55,854)
CERN Archives (51,334) Pauli Archives (3,807) DSU Archives (713)

Search also:

☐ CERN Indico
☐ KISS Preprints
☐ SPIRES HEP

Illustration 1: The collections present in the CERN Document Server

3.4 Detailed Specifications

3.4.1 Technological restrictions

As stated in the INSTALL file that ships with the software, INVENIO currently requires the following setup:

- A Unix-like operating system. The main development and production platforms for INVENIO at CERN are the GNU/Linux distributions Debian, Gentoo, Scientific Linux (aka RHEL) and Ubuntu.
- MySQL versions 4.1 or 5.0 .
- Python v2.4 or above.
- Apache 2 server, with support for loading DSO modules
- mod_wsgi Apache module.
- HTML, CSS, JavaScript and jQuery for the web pages.
- Some C and Lisp are used in INVENIO as well, which requires the use of Makefiles and a make-install work-flow for each change to the Python source files as well.

It is interesting to note here as well that INVENIO uses a mostly procedural / imperative programming style and only rarely an object-oriented design.

3.4.2 Overview of the present INVENIO modules

At the outset of this project, this is what the overview of the various INVENIO modules looked like (Illustration 2):

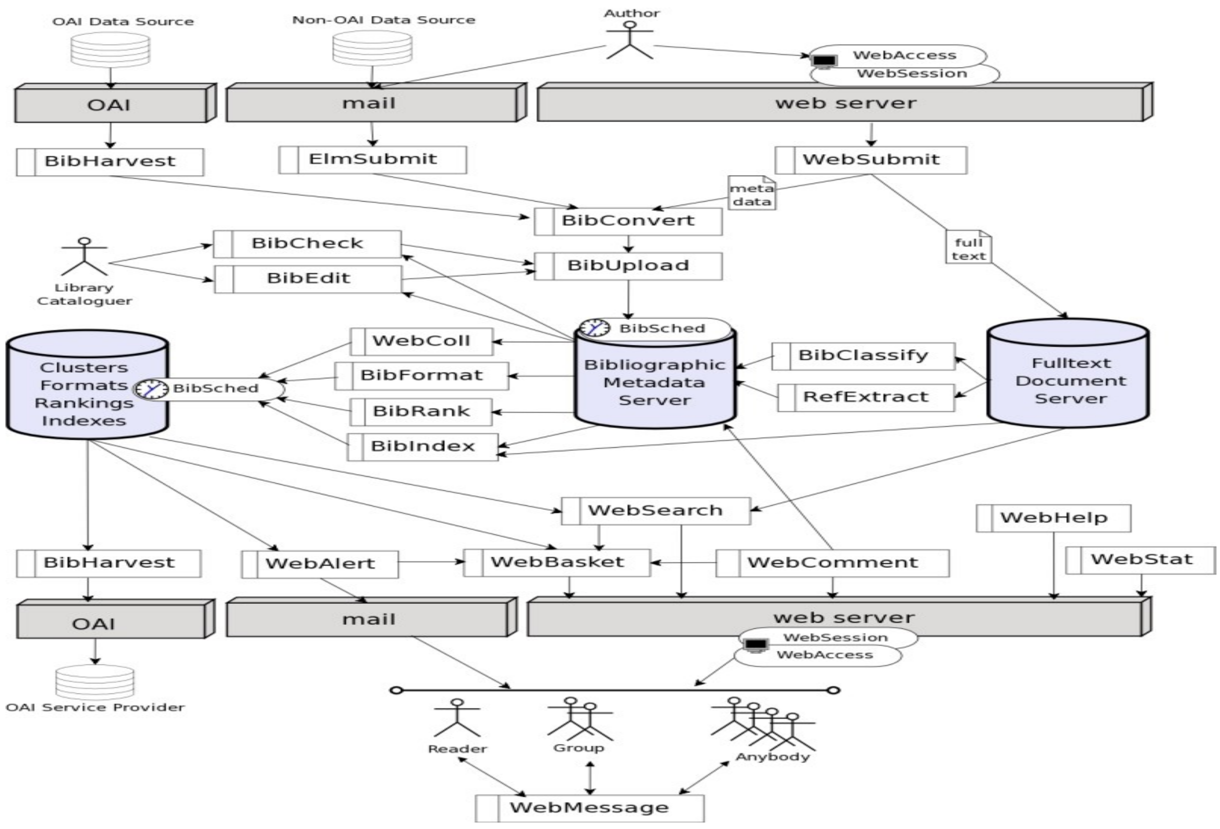


Illustration 2: Overview of the INVENIO modules

It was recommended to me by members of the CDS team to put only a minimal amount of code into a new BibAuthority module, while directly integrating most of it into the code of the existing modules. Future refactoring would be possible to migrate that code into the BibAuthority module at a later time. As it currently stands, there is an additional BibAuthority module containing only a bibauthority_config.py file as well as a bibauthority_engine.py file containing some purely authority-related functions. The rest of the code of this project lives in the respective modules that needed to be adapted for authority control to work: BibEdit for the AJAX calls and drop-down functionality, BibIndex for the Indexing of Data coming from linked authority records, BibFormat for the output formatting of authority records, and WebSearch for special authority-related search-options.

3.4.3 Types of authority records

For this project, we will deal with only 4 types of authority records: Authors, Institutions, Journals and Subjects. However, these 4 types shall not be hard-coded into the system, but should rather be configurable variables so that further types can easily be added by system administrators in the future without any knowledge of Python.

3.4.4 The editing and storage of authority records

Authority records need to be editable like any other MARC record in INVENIO. The advantage of using authority records is that they share the same basic MARC structure with the bibliographic records already stored and edited by INVENIO. This means that simple back-end uploading, storage and even editing of authority records will work out of the box for administrators of INVENIO. In a first stage, non-admin users will not need to upload or edit authority records.

3.4.5 Enriching the searches with data from authority records

When a user searches for publications e.g. of a particular author, he/she may not spell the author's name the same way it appeared in the publication. In order to return successful search results, the indexing needs to take into account all the variations of how to write an author's name. This can be done through the use of authority records, where the indexer not only indexes the author name as indicated in the bibliographic record, but also indexes alternative spellings or names as contained in the referenced authority record. In addition, the search needs to provide ways of disambiguating between identical search terms with different significance. Authority records can provide the search index with additional information about the searched term that will allow the user to pin down the exact thing he or she is searching for.

The way INVENIO's search function works, the indexes need to be updated whenever there is a change to one of the records. This requires some special care when we introduce authority records, since a change in an authority record requires not only updating the index for this individual record, but also all of the indexes for the bibliographic records that make use of this authority record.

3.4.6 Cross-referencing between MARC records

For authority records to be more than a simple collection of logically related data, we need a way for one MARC record to reference another. An example for this would be between a bibliographic record and an authority record, where the bibliographic record references the authority record. E.g., a bibliographic record could have a particular author working for a particular institution. In stead of (or in addition to) simply writing the name of this institution, we want the bibliographic record to contain some kind of reference to the actual authority record of the institution in question. Only in this way can we intelligently make use of the authority record data for the bibliographic record as well. MARC 21 has defined the \$0 subfield for this purpose[12]. This subfield is called the “Authority record control number” or “standard number” and will in our case contain an alphanumeric string that will uniquely identify the referenced authority record.

3.4.7 Validation of authority-controlled fields during the uploading process

When an admin uploads a new record to INVENIO, he/she needs to be able to choose whether or not the uploader should check for “broken links” within the references to authority records. If such broken links are found, the user needs to receive a feed-back explaining which records are affected. If the CLI is used directly for uploading new records, the feed-back will be in form of a message in the terminal. If the upload job is created dynamically by a python script (e.g. after the use of BibEdit or WebSubmit), then these messages will be written to the respective log files.

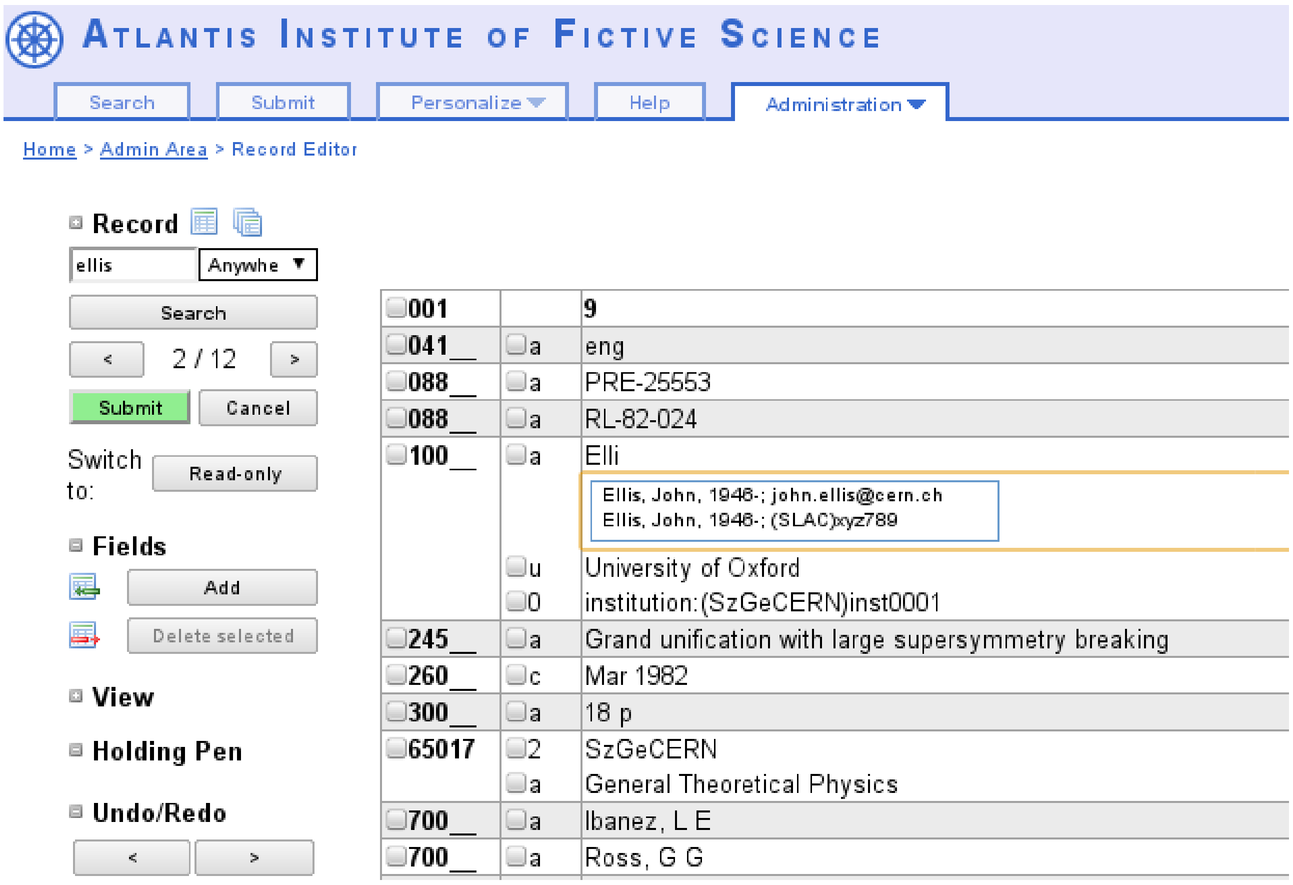
12 http://www.loc.gov/marc/bibliographic/ecbdcntf.html

3.4.8 Auto-suggestion and validation for authority controlled fields in the on-line editing process

In order to facilitate the use of authority records for the library catalogers, certain fields of the bibliographic records will offer auto-complete (called "auto-suggest" in INVENIO) functionalities through the use of drop-down menus, based on what the user types into the given field. These drop-down field will offer the same search features that are available for the global INVENIO search as well. This means that the librarian can type in an alternative spelling for an authority (e.g. for an author), and the drop-down will show the main names for which this alternative writing is recorded in an authority file. In addition, the admin will have the choice whether to configure INVENIO to allow only values from authority records or whether he/she can also enter values manually, without a correspondence in the drop-down suggestions.

In case the drop-down contains multiple values with the same writing, additional data can be displayed in the drop-down to disambiguate, data that will not be carried over into the MARC subfield upon selection however.

Illustration 3 below shows an example of how such an auto-suggest drop-down list will look.



001		9
041__	a	eng
088__	a	PRE-25553
088__	a	RL-82-024
100__	a	Elli Ellis, John, 1946-; john.ellis@cern.ch Ellis, John, 1946-; (SLAC)xyz789
	u	University of Oxford
	0	institution:(SzGeCERN)inst0001
245__	a	Grand unification with large supersymmetry breaking
260__	c	Mar 1982
300__	a	18 p
65017	2	SzGeCERN
	a	General Theoretical Physics
700__	a	Ibanez, L E
700__	a	Ross, G G

Illustration 3: An example of the auto-suggest feature for the author's name

3.4.9 Periodic consistency checks for the already present data

The BibAuthority module needs a CLI command that allows checking for consistency in the references towards authority records. This means scanning through all the bibliographic MARC fields that have been specified by the admin as being under authority control, and then verifying that the authority record control number in their $0 subfields corresponds to an existing authority record containing that same number as a standard identifier.

We need to make this check periodic with the scheduler that INVENIO offers.

3.4.10 New output formats for authority records

If a librarian or a regular user wishes to view an authority record, this should be possible with the same types of HTML formats available for bibliographic records.

4 Design + Implementation

4.1 Introduction

The following section describes how the use cases described above will need to be built into new or existing functions for INVENIO. For this to be successful, it was necessary to plan the design of these functions in regard to the already existing INVENIO architecture. Since Design and Implementation cannot be as strictly separated in this project as in other projects, I have decided to put their description into the same section. Nevertheless, for each point I will focus first of all on the more conceptual aspect of the newly created functions, and only at the end of each chapter will I talk about the actual adaptations to the existing code that were necessary for the implementation.

4.2 The editing and storage of authority records

As mentioned in the previous section, MARC 21 authority records have the same internal structure as bibliographic records. From the point of view of INVENIO, both bibliographic as well as authority records are simply MARC. As long as it is valid MARC it can upload, store or delete any record presented to it and offers the same tools to the admin for editing the records as well. This turned out to be a very nice feature for the authority control project since it meant that much of INVENIO could simply be reused as is for the storage and editing of authority records. That said, it should be noted however that whereas INVENIO is agnostic about the type of MARC record it is presented on a pure MARC level, as soon as one gets into the functionalities of how INVENIO presents this data to the user in a more user-friendly form, the differences are in deed no longer negligible. This is primarily due to the fact that identical field numbers of an authority record can have quite a different meaning from the same field in a bibliographic record (cf. The chapter on MARC 21 in the Analysis section)

4.3 Enriching the searches with data from authority records

4.3.1 Introduction

In the previous section, we described how certain subfields from authority records can be used to enrich the indexed data of related subfields in bibliographic records. In this section, we will describe in detail how we must go about to do this for INVENIO.

There are two cases that need special attention for this feature when considering the links between bibliographic and authority records. The first case is relatively simple and requires the enriching of bibliographic data with data from authority records whenever a bibliographic record is being indexed. The second is a bit more complex, for it requires detecting which bibliographic records should be re-indexed, based on referenced authority records having been updated within a given date range.

4.3.2 Indexing by record ID, by modification date or by index type

First of all, we need to say something about how INVENIO let's the admin index the data. INVENIO's indexer (BibIndex) is always run as a task that is executed by INVENIO's scheduler (BibSched). Typically, this is done either by scheduling a bibindex task from the command line (manually), or it is part of a periodic task (BibTask) run directly from BibSched, typically ever 5 minutes. In case it is run manually, the user has the option of specifying certain record IDs to be re-indexed, e.g. by specifying ranges of IDs or collections to be re-indexed. In this case, the selected records are re-indexed whether or not there were any modifications to the data. Alternatively, the user can specify a date range, in which case the indexer will search all the record IDs that have been modified in the selected date range (by default, the date range would specify all IDs modified since the last time the indexer was run) and update the index only for those records. As a third option, the user can specify specific types of indexes. INVENIO lets you search by different criteria (e.g. 'any field', 'title', 'author', 'abstract', 'keyword', 'journal', 'year', 'fulltext', …), and each of these criteria corresponds to a separate index, indexing only the data from the relevant MARC subfields. Normally, the indexer would update all index types for any given record ID, but with this third option, the user can limit the re-indexing to only specific types of indexes if desired.

Note: In reality, INVENIO creates not only 1 but 6 different indexes per index type. 3 are forward indexes (mapping words, pairs or phrases to record IDs), 3 are reverse indexes (mapping record IDs to words, pairs or phrases). The word, pair and phrase indexes are used for optimizing the searching speed depending on whether the user searches for words, sub-phrases or entire phrases. These details are however not relevant for BibAuthority. It simply finds the values to be indexed and passes them on to the indexer which indexes them as if it was data coming directly from the bibliographic record.

4.3.3 Enriching the index data – simple case

Once the indexer knows which record ID (and optionally, which index type) to re-index, including authority data is simply a question of checking whether the MARC subfields currently being indexed are under authority control (as specified in the BibAuthority configuration file). If they are, the indexer must follow the following (pseudo-)algorithm which will fetch the necessary data from the referenced authority records:

```
For each subfield and each record ID currently being re-indexed:
    If the subfield is under authority control (→ config file):
        Get the type of referenced authority record expected for this field
        For each authority record control number found in the corresponding
        'XXX__0' subfields and matching the expected authority record type (control
        number prefix):
            Find the authority record ID (MARC field '001' control number)
            corresponding to the authority record control number (as contained in
            MARC field '035' of the authority record)
            For each authority record subfield marked as index relevant for the
            given $type (→ config file)
                Add the values of these subfields to the list of values to be
                returned and used for enriching the indexed strings.
```

The strings collected with this algorithm are simply added to the strings already found by the indexer in the regular bibliographic record MARC data. Once all the strings are collected, the indexer goes on with the usual operation, parsing them 3 different times, once for phrases, once for word-pairs, once for words, which are used to populate the 6 forward and reverse index tables in the database.

4.3.4 Updating the index by date range

When a bibindex task is created by date range, we are presented with a more tricky situation which requires a more complex treatment for it to work properly. As long as the bibindex task is configured to index by record ID, the simple algorithm described above is enough to properly index the authority data along with the data from bibliographic records. This is true also if we use the third option described above, specifying the particular index type to re-index with the bibindex task. However, if we launch a bibindex task based on a date range (by default the date range covers the time since the last time bibindex task was run on for each of the index types), bibindex would have no way to know that it must update the index for a specific bibliographic record if one of the authority records it references was modified in the specified date range. This would lead to incomplete indexes.

A first idea was to modify the time-stamp for any bibliographic records as soon as an authority record is modified. Every MARC record in INVENIO has a 'modification_date' time-stamp which indicates to the indexer when this record was last modified. If we search for dependent bibliographic records every time we modify an authority record, and if we then update the 'modification_date' time-stamp for each of these dependent bibliographic records, then we can be sure that the indexer would find and re-index these bibliographic records as well when indexing by a specified date-range. The problem with this is a performance problem. If we update the time-stamp for the bibliographic record, this record will be re-indexed for all of the mentioned index-types ('author', 'abstract', 'fulltext', etc.), even though many of them may not cover MARC subfields that are under authority control, and hence re-indexing them because of a change in an authority record would be quite useless. In an INVENIO installation there would typically be 15-30 index-types. Imagine if you make a change to a 'journal' authority record and only 1 out of the 20+ index-types is for 'journal'. INVENIO would be re-indexing 20+ index types in stead of only the 1 index type which is relevant to the the type of the changed authority record.

There are two approaches that could solve this problem equally well. The first approach would require checking – for each authority record ID which is to be re-indexed – whether there are any dependent bibliographic records that need to be re-indexed as well. If done in the right manner, this approach would only re-index the necessary index types that can contain information from referenced authority records, and the user could specify the index type to be re-indexed and the right bibliographic records would still be found. The second approach works the other way around. In stead of waiting until we find a recently modified authority record, and then looking for dependent bibliographic records, we directly launch a search for bibliographic records containing links to recently updated authority records and add the record IDs found in this way to the list of record IDs that need to be re-indexed.

Of the two approaches, the second one was choses based solely upon considerations of integration into existing INVENIO code. As indexing in INVENIO currently works, it is more natural and easily readable to apply the second method than the first.

According to the second method, the pseudo-algorithm for finding the bibliographic record IDs that need to be updated based upon recently modified authority records in a given date range looks like this:

```
For each index-type to re-index:
```

```
For each subfield concerned by the index-type:
    If the subfield is under authority control (→ config file):
        Get the type of authority record associated with this field
        Get all of the record IDs for authority records updated in the
        specified date range.
        For each record ID
            Get the authority record control numbers of this record ID
            For each authority record control number
                Search for and add the record IDs of bibliographic
                records containing this control number (with type in the
                prefix) in the 'XXX__0' field of the current subfield to
                the list of record IDs to be returned to the caller to be
                marked as needing re-indexing.
```

The record IDs returned in this way are added to the record IDs that need to be re-indexed (by date range) and then the rest of the indexing can run as usual.

4.3.5 Implementation specifics

The pseudo-algorithms described above were used as described in this document, but were not each implemented in a single function. In order for parts of them to be reusable and also for the various parts to be properly integrated into existing python modules with similar functionality (e.g auxiliary search functions were added to INVENIO's search_engine.py code), the pseudo-algorithms were split up into multiple nested function calls and integrated where it seemed to best fit the existing code base of INVENIO. In the case of the pseudo-algorithm described in "Updating the index by date range", the very choice of the algorithm had already depended on how to best integrate it into the existing code for date-range related indexing. It should be mentioned here as well, that the design and implementation of these functions was done in agreement with Tibor Simko, the head architect and developer, and therefore depends to a large degree also on his suggestions and preferences.

There are a few possible improvements that could be made to the current code, but these would require refactoring a substantial amount of the existing code, a task that I was explicitly not required to do during this project. One of them concerns the way the indexing by date-range first indexes the bibliographic records modified within the given date-range, then indexes the bibliographic records that have an authorID entry that was modified within that given date-range (a separate module not related to my project), and then finally those that have a referenced authority record modified within that given date-range. The advantage of this is that in the bibindex task logs, we see which bibliographic records were re-indexed for which of the 3 reasons (because they are treated separately). The disadvantage is, that the same bibliographic record might be re-indexed 2-3 times, depending on which of the 3 cases just described find that it needs to be re-indexed for the given date range.

In another place, I ended up refactoring the existing code, based on a recommendation by Tibor Simko for code clarity. However, as I realized during the refactoring process, these changes would have undone an important MySQL query optimization (querying by range with “value BETWEEN x AND y” in stead of one query per value), and so a different solution needed to be found. We finally found a solution that was able to combine both readability and the preservation of the MySQL query optimization.

4.4 Cross-referencing between MARC records

In the previous section we saw the need for alphanumeric strings stored in the $0 subfields of fields with other, authority-controlled subfields. The format of these alphanumeric strings for INVENIO will in part be determined by the MARC standard itself, which states that:

```
Subfield $0 contains the system control number of the related authority
record, or a standard identifier such as an International Standard Name
Identifier (ISNI). The control number or identifier is preceded by the
appropriate MARC Organization code (for a related authority record) or
the Standard Identifier source code (for a standard identifier scheme),
enclosed in parentheses. See MARC Code List for Organizations for a
listing of organization codes and Standard Identifier Source Codes for
code systems for standard identifiers. Subfield $0 is repeatable for
different control numbers or identifiers.[13]
```

An example of such a string could be “(SzGeCERN)abc1234”, where “SzGeCERN” would be the MARC organization code[14], and abc1234 would be the unique identifier for this authority record within the given organization.
Since it is possible for a single field (e.g. field '100') to have multiple $0 subfields for the same field entry, we need a way to specify which $0 subfield reference is associated with which other subfield of the same field entry.

For example, imagine that in bibliographic records both '700__a' ('other author' *name*) as well as '700__u' ('other author' *affiliation*) are under authority control. In this case we would have two '700__0' subfields. Of of them would reference the *author* authority record (for the *name*), the other one would reference an *institution* authority record (for the *affiliation*). INVENIO needs some way to know which $0 subfield is associated with the $a subfield and which one with the $u subfield.

We have chosen to solve this in the following way. Every $0 subfield value will not only contain the authority record control number, but in addition will be prefixed by the type of authority record (e.g. 'AUTHOR', 'INSTITUTION', 'JOURNAL' or 'SUBJECT), separated from the control number by a ':'. A possible $0 subfield value could therefore be: “author:(SzGeCERN)abc1234”. This will allow INVENIO to know that the $0 subfield containing “author:(SzGeCERN)abc1234” is associated with the $a subfield (author's name), containing e.g. “Ellis, John”, whereas the $0 subfield containing “institution:(SzGeCERN)xyz4321” is associated with the $u subfield (author's affiliation/institution) of the same field entry, containing e.g. “CERN”.

4.5 Upload validation and periodic consistency checks

What were mentioned as distinct tasks in the scope statement are in fact closely related. Both the

13 http://www.loc.gov/marc/bibliographic/ecbdcntf.html
14 cf. http://www.loc.gov/marc/organizations/org-search.php

validation during the upload procedure as well as the consistency checks need to verify if authority record control numbers contained within $0 subfields of bibliographic records correspond to an existing authority record in INVENIO. In addition, this functionality can check for the authority record type at the same time, comparing the prefix of the $0 value with the '980' field values of the referenced authority record, making sure it is present.

In order to unite the two uses for this functionality, a function will be created which can be called in both cases. This function receives the authority record control number prefixed with the authority record type (alphanumeric string) and returns a dictionary containing the relevant consistency information to be returned. The dictionary contains a key 'single_record_exists' and another key 'has_right_type', both of which map to a boolean value that indicates whether a) a single authority record exists with the given authority record control number, and b) whether this authority record contains a '980__a' subfield with the given authority record type. Depending on whether a), b) or both are true, INVENIO will be able to tell the user some useful information about the validity of the MARC data being uploaded or about the consistency of the record database in general.

This function needs to be wrapped of course by other INVENIO functions / CLI commands. The upload validation requires a function that can be integrated into the existing BibUpload module, giving the user a warning if a broken reference to an authority record was found, whereas the periodic consistency check must be run as a part of a task that can be run by the scheduler (BibSched). For the latter, since there is currently no consistency checker available in the public INVENIO code, my project will offer a separate checker task as part of the BibAuthority module. There is currently a BibCheck module in development (written in Lisp), but it is not yet available do other developers. The code should be simple enough to easily be integrated into this BibCheck module, if one day it becomes an integrated part of INVENIO.

4.6 Auto-suggestion and validation for authority controlled fields in the on-line editing process

In the previous section we described how a librarian will be offered a drop-down with suggestions based upon the characters typed into an authority-controlled subfield of the web-based editing tool (BibEdit). In this section we will take a closer look at how BibEdit currently works, and which steps are necessary for INVENIO to integrate authority control into the current system.

4.6.1 Entering data

When the librarian starts typing data on the BibEdit web-interface into a MARC subfield that is under authority-control, there will be a JavaScript event-listener attached to this input field. This event-listener has the following behavior. The first 3 characters won't have any visible effect for the user. The auto-complete behavior only kicks in after the 4th character has been typed, allowing an additional timeout of a configurable time (typically between 100 and 250 ms) before the characters in the input field are sent via AJAX to the server.

4.6.2 AJAX request: back-end

Once a few characters have been sent to the server, the server will run a search based on this string and based on the index type to be used for this search, as defined in the BibAuthority configuration file (cf. Admin User Manual for details on how to configure BibAuthority for Indexing and Searching). Once the matching authority files have been found, both the authority record control number as well as the actual value to be inserted into the field that triggered the AJAX request (again, with the help of the configuration file). In a final step, the value for the drop-down list needs to be calculated (cf. the next point on JSON formatting).

4.6.3 AJAX response: JSON format

Another thing to consider is the format of the data the server returns via AJAX to the browser, in order to populate the drop-down list.

Example: The user types 'Elli'. The AJAX call returns a JSON list of suggestions in the form of dict objects like this:

```
[
     {
             'show'         : 'Ellis, John, 1946-, john.ellis@cern.ch',
             'insert_here'  : 'Ellis, John',
             'insert_0'     : 'author:(SzGeCERN)abc123',
     },
     {
             'show'         : 'Ellis, John, 1946-, author:(SLAC)xyz789',
             'insert_here'  : 'Ellis, John',
             'insert_0'     : 'author:(SLAC)xyz789',
     },
     {
             … some more auto-suggest data …
     },
]
```

The drop-down list would contain the 'show' values ("Ellis, John, 1946-, john.ellis@cern.ch" and "Ellis, John, 1946-, author:(SLAC)xyz789"). If the user chooses the first suggestion, INVENIO would insert 'author:(SzGeCERN)abc123' into the $0 subfield of the field currently being edited (e.g. '100__0'), and "Ellis, John" directly into the subfield currently being edited (e.g. '100__a').

The format of the 'show' value can be configured by the admin in bibauthority_config.py (cf. the document "Admin User manual") In case multiple suggestions have have the same 'insert_here' value, but not all of the suggestions have data in the additional disambiguation fields, INVENIO will automatically add the authority record control number to the end of the 'show' value to guarantee a unique value for disambiguation purposes. In the case above, the author's date and the author's email were chosen by the admin as disambiguation fields, but because the dates were identical for two different authority records, and because the second one didn't have any e-mail data, the authority record control number ("author:(SLAC)xyz789") was used instead.

The algorithm used to create the 'show' string is written as pseudo-code below:

Create one *show string* per *authority record,* consisting initially of the value of each authority record's 'insert_here_field'.

Create a list of *similar authority records* containing the same *show string*.

Create a list of *remaining fields* containing a copy the list of 'disambiguation_fields'.

While the list of *similar authority records* is not empty AND the list of *remaining fields* is not empty:

```
        For each authority record in the list of authority records:
            Remove the next disambiguation field from remaining fields.
            Get the disambiguation string value from the disambiguation field of the
            current authority record.
            If the disambiguation string is empty:
                Replace it with this authority record's the authority record control
                number.
            Add the disambiguation string to the end of the show string for this authority
            record, separated by a comma.
        Replace the list of similar authority records with a new list of similar authority
        records that have the same show string.
    If the list of similar authority records is not empty:
        Add the authority record's authority record control number to the end of the show
        string for each authority record in the list of similar authority records, separated by a
        comma.
```

4.6.4 Creating the drop-down list

Furthermore, we must consider is the drop-down list that appears when a librarian starts typing characters into an authority-controlled subfield. INVENIO tries to use jQuery and jQuery UI wherever possible. jQuery UI has an auto-complete plug-in which matches this use case very well. I can easily be made to work with the BibEdit pages. Furthermore, one of the INVENIO developers in the SLAC team at Stanford (Joe Blaylock), has already implemented a similar functionality for the INSPIRE site that uses INVENIO. His code and help was of great help in integrating this to INVENIO.

4.6.5 Inserting all the data

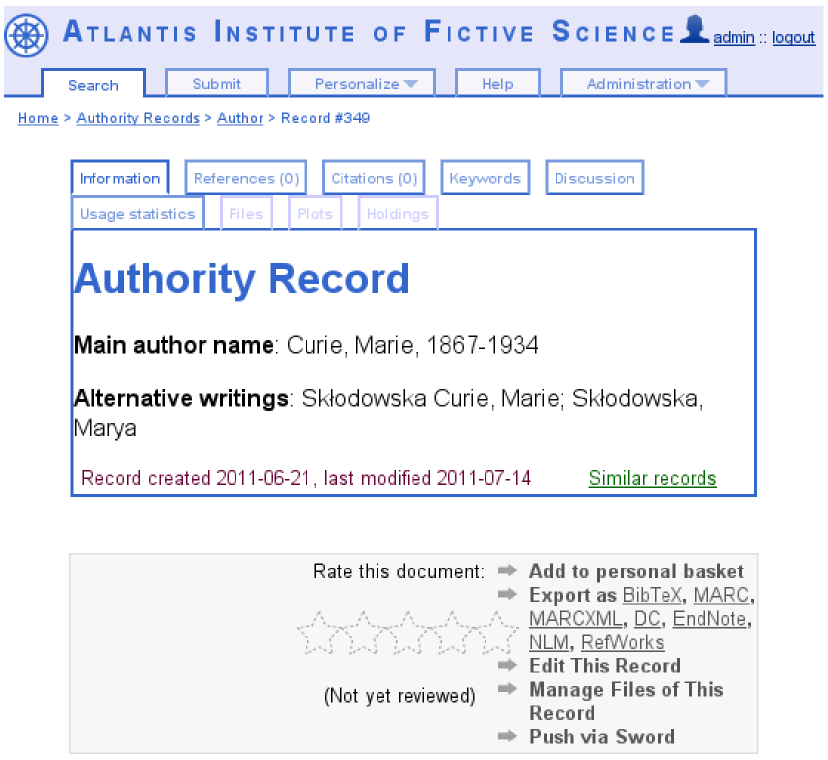
The next question to be addresses is what happens when the user selects one of the entries from the drop-down list. If the user chooses e.g. the second one from the drop-down, the 100__a field will be filled with "Ellis, John" and the '100__0' field will be filled with "author:(SLAC)xyz789". This way, the librarian can type the human readable form and the system will automatically complete the MARC data with the alphanumeric string referring to the authority record.

My meeting with Javier Martin Montull was helpful to get to know the existing JavaScript library in bibedit_engine.js which can to a great part be reused for the dynamic insertion of new fields. This way all of the AJAX calls to the server, the synchronization across multiple browsers (to prevent concurrent editing of the same file on multiple browsers) are kept in tact.

4.7 New output formats for authority records

Creating new output formats for authority records required getting familiar with yet another INVENIO module, BibFormat. This module offers a special syntax (similar to HTML) for defining format templates, tied to simple python functions that construct the building blocks to be included in these files. Since authority records use the same basic MARC format, all of the MARC related output formats that INVENIO offers like MARC and MARCXML worked out of the box. For more direct and human-readable access to the contents of these files however, we needed at least one "detailed" and one "brief" HTML output format for displaying in the browser.

Illustration 4 shows the result of the newly created "detailed" HTML output format for authority records displaying a simple authority record containing only the various names as well as the dates of birth and death of "Marie Curie".



Authority Record

Main author name: Curie, Marie, 1867-1934

Alternative writings: Skłodowska Curie, Marie; Skłodowska, Marya

Record created 2011-06-21, last modified 2011-07-14

Illustration 4: A (detailed) HTML output format for a simple authority record

5 Tests

INVENIO uses 3 types of tests to help developers identify existing bugs as INVENIO evolves. These are Unit Tests, Regression Tests and Web Browser Tests. The first two are written in Python, the last type is created with “Selenium”, a web browser testing tool.

At the time of this report, unit tests are the only type of test created and run for the new BibAuthority code. The remaining weeks at CERN will allow the creation of full regression and web browser tests as well.

Authority record mock data was added to the miscutil/demobibdata.xml file to enable an easy and clean setup of INVENIO with demo data to test with.

6 Possible Improvements

6.1 General

Authority control was integrated and implemented for the core INVENIO modules, so that a solid base was available for other developers who wish to add authority related functionalities to their modules. Therefore most of the modules that haven't been touched by this bachelor project can still be updated to add authority-specific functionalities.

6.2 Syntax-error immune configuration file

Perhaps the one clearly unfinished task of this project is the format of the configuration file. The INVENIO admin will typically not be a Python programmer and should not have to use a Python specific syntax for configuring authority control in INVENIO. Furthermore, any syntax errors made in the bibauthority_config.py configuration file could potentially break the proper functioning of INVENIO all-together. This was simply a task that could not be finished by the hand-in date of this report. In the remaining time of the internship at CERN, this will be changed so that the admin can use a simpler notation syntax and syntax errors will fail gracefully.

7 Conclusion

7.1 General

This bachelor project was a unique experience for me. Besides diving into a new project, teaching me Python and advanced features of GIT, introducing me into the world of digital libraries, MARC and search engine indexing, it also granted me the chance to work at a large and renowned international organization in a region of Switzerland and France previously unknown to me. I am satisfied with the work I was able to do in the relatively short amount of time available. My work allowed me to meet the highly diverse and bright international personnel of CERN, many of them students like myself, as well as collaborating with users and contributers to INVENIO from German and California. I felt lucky to work in a very friendly team and office environment. In my free hours, I tried to discover as much of CERN as possible, without having any of the organized presentations normally offered to "summer students".

7.2 Lessons learned

7.2.1 GIT

Although I had already worked with GIT on a previous project, my time at CERN made me realize that I had only scratched the surface of this powerful tool. INVENIO development usually follows a rather elaborate GIT Work-flow[15] where developers never work on the master branch, but instead always work on branches dedicated to the specific feature they are working on, which is then pushed to their public repository as a branch. During the work on a particular branch, recent changes to the public master branch (made by other developers) can be integrated into the local branch by a GIT function called "rebasing", where the original master base of the branch is swapped for the new version of master. Also, before pushing a finished feature branch to the public repository, another feature of GIT, called "squashing", is used to clean up the commit history, so that other developers will see the branch modifications as a single commit. The Head Developer then takes care of merging the new public branches back into the public master (or, in some cases, into public branches of previous versions of INVENIO or into the "next" branch for future public versions). For local development, GIT offers all of the usual functionalities of a versioning system like SVN, i.e. to switch quickly between branches, restore individual files from the commit history, browse the commit history, view the changes between individual commits, add additional changes to existing commits, etc. In brief, I learned to what extent GIT is a very flexible and powerful tool for collaborative editing and keeping track of numerous lines of development on multiple versions of the same software.

7.2.2 Python

Before coming to CERN I had never worked with Python. A previous CERN intern had told me that it was very similar to Java, so I didn't expect it to be a difficult transition. Although the transition was in deed not very difficult, I disagree with the simple comparison with Java. I found Python to be quite a bit different in many ways. Python does not offer all of the features typically associated with object-oriented programming (there is no encapsulation, e.g.). Contrary to Java, Python allows

15 https://twiki.cern.ch/twiki/bin/view/CDS/GitWorkflow

not only object-oriented programming, but also procedural/imperative programming and to a certain degree even functional programming, all within the same code. Python is interpreted, not compiled, which means it can be used for rapid prototyping directly within a python interpreter (e.g. iPython).

7.3 Problems encountered

7.3.1 Debugging INVENIO

Despite the advantages of Python mentioned above, INVENIO also uses C and Lisp for performance, and because of this we loose many of the benefits of an interpreted language, since any change made to the actual INVENIO python files needs to be run through a “make install” process, which copies the python files to a second location from where they are run by apache. This made debugging INVENIO quite a bit slower than it could be with a regular step-through debugger. I met only one INVENIO developer (in another team) who followed neither the make-install nor the GIT work-flow officially suggested for INVENIO. This allowed him to run and debug the source files directly from within Eclipse. Due to the short duration of my project, however, I decided to stick with the work-flows used in my team, since I would find more support this way in case of encountered problems and wouldn't loose time creating a completely new setup of INVENIO.

7.3.2 Project management

Perhaps the most challenging aspect of my project was not of a technical nature, but rather one of project management and coordination with my superiors. As mentioned in the introduction, I didn't have a clear project description until the end of the 2nd week of my project. Any attempt to speed up the process was fruitless, and an explicit mention of the problem was encountered with an equally explicit statement that I would have to live with this. At the same time, an attempt to ask for a later hand-in date for my bachelor project report was turned down for nontransparent administrative reasons by the EIA-FR, despite the fact that my internship at CERN was in all relevant aspects fully identical with projects done “abroad” where the students can hand in their projects as much as 5 weeks later than those done in Switzerland. Basically, I found myself “serving two masters” who were both unwilling to make concessions for the dead-lines and restrictions imposed by the other party. This left me with the task of trying to use my time as efficiently as possible so that I could implement and document a maximum of features by the end of the 7 weeks allowed by the EIA-FR. I hope this document, despite the short time at hand, manages to show the fruits of this effort.

8 Illustration Index

9 Glossary

Term	**Meaning**
Admin	The INVENIO administrator who installs and configures the INVENIO software for the librarian.
Authority record	A record describing an authority (author, institution, journal, other heading, subject, ...)
Bibliographic record	A record describing a publication (article, book, photo, video, …)
Bib* (e.g. BibEdit)	For the meaning of the INVENIO modules, cf. http://invenio-demo.cern.ch/help/hacking/modules-overview
CLI	Command-line interface, Terminal
Library Cataloger	The librarian who uses INVENIO to edit the catalog information for the digital library.
MARC	Machine Readable Cataloging. A set of standards for the representation and communication of bibliographic and related information in machine-readable form, and related documentation.
MARC 21	MARC 21 was designed to redefine the original MARC record format for the 21st century and to make it more accessible to the international community.
Web* (e.g. WebSearch)	For the meaning of the INVENIO modules, cf. http://invenio-demo.cern.ch/help/hacking/modules-overview

10 Contents of the CD-ROM

The following table lists the content of the CD-ROM delivered with the final report of this bachelor project.

Path	**Contents**
/**README.pdf**	Lists the contents of this CD
/**code**/	The INVENIO source code, including BibAuthority
/code/**INSTALL**	The installation manual for INVENIO
/code/**modules**/	The INVENIO Python modules
/**doc**/	All the documentation of this project
/doc/**final_report.pdf**	The final report of this project
/doc/**flyer.pdf**	The flyer describing this bachelor project
/doc/**manual_admin.pdf**	The User Manual for INVENIO Administrators
/doc/**manual_librarian.pdf**	The User Manual for Librarians

11 Appendices

11.1 Librarian's User Manual

The librarian's user manual explains how to MARC authority records, how to create collections of authority records, and how to use authority record data within BibEdit to edit bibliographic records.

11.2 Admin User Manual

The Administrator user manual explains to the INVENIO administrator how to configure INVENIO to work with authority records for search indexing and BibEdit auto-suggestion drop-down lists.

11.3 Scope Statement

The scope statement for this project has been appended to this report for reference.

11.4 Planning

The planning for this project has been appended to the end of this document.

11.5 List of meetings

There were quite a few meetings involved in this project in order to coordinate the work on BibAuthority with the various developers working on the various INVENIO modules, some of them in other countries (visiting Geneva during my internship), as well as with librarians and INVENIO admins who are interested in authority control for their own uses.

11.6 Selected meeting notes

The meeting notes that seemed relevant for this report were appended to the end of this document.

Admin User Manual

Introduction

The INVENIO admin can configure the various ways in which authority control works for INVENIO by means of the bibauthority_config.py file. The location and full contents of this configuration file with a commented example configuration are shown at the end of this document. Their functionality is explained in the following chapters.

Enforcing types of authority records

INVENIO is originally agnostic about the types of authority records it contains. Everything it needs to know about authority records comes, on the one hand, from the authority record types that are contained within the '980__a' fields, and from the configurations related to these types on the other hand. Whereas the '980__a' values are usually edited by the librarians, the INVENIO configuration is the responsibility of the administrator. It is important for librarians and administrators to communicate the exact authority record types as well as the desired functionality relative to the types for the various INVENIO modules.

BibEdit

For examples authority control works in BibEdit from a user's perspective, cf. the "Librarian's User Manual for BibEdit".

As admin of an INVENIO instance, you have the possibility of configuring which fields are under authority control. In the "Configuration File Overview" at the end of this document is an example of a configuration which will trigger an auto-complete functionality for the '100__a', '100__u', '110__a', '130__a', '150__a', '700__a' and '700__u' fields of a bibliographic record in BibEdit. The keys of the "CFG BIBAUTHORITY CONTROLLED FIELDS" dictionary indicate which bibliographic fields are under authority control. These fields will automatically propose an auto-complete dropdown list in BibEdit, although the user can still enter values manually without use of the drop-down list. The values associated with each key of the dictionary indicate which kind of authority record is to be associated with this field. In the example given, the '100__a' field is associated with the authority record type 'AUTHOR'.

The "CFG BIBAUTHORITY AUTOSUGGEST OPTIONS" dictionary gives us the remaining configurations, specific only to the auto-suggest functionality. The value for the 'index' key determines which index type will be used find the authority records that will populate the drop-down with a list of suggestions (cf. the following chapter on configuring the BibIndex for authority records). The value of the 'insert_here_field' determines which authority record field contains the value that should be used both for constructing the strings of the entries in the drop-down list as well as the value to be inserted directly into the edited subfield if the user clicks on one of the drop-down entries. The value of the 'sort' key tells INVENIO how to sort the entries in the drop-down list. If a popularity sort is chosen, the drop-down entries will be sorted according to how often the associated authority record is referenced in this particular INVENIO instance. Finally, the value for the 'disambiguation_fields' key is an ordered list of authority record fields that are used, in the order in which they appear in the list, to disambiguate between authority records with exactly the same value in their 'insert_here_field'.

BibIndex

As an admin of INVENIO, you have the possibility of configuring how indexing works in regards to authority records that are referenced by bibliographic records. When a bibliographic record is indexed for a particular index type, and if that index type contains MARC fields which are under authority control in this particular INVENIO instance (as configured by the, "CFG BIBAUTHORITY CONTROLLED FIELDS" dictionary in the bibauthority_config.py configuration file, mentioned above), then the indexer will include authority record data from specific MARC fields of these authority records in the same index. Which authority record fields are to be used to enrich the indexes for bibliographic records can be configured by the "CFG BIBAUTHORITY AUTHORITY SUBFIELDS TO INDEX" dictionary. In the example below each of the 4 authority record types ('AUTHOR', 'INSTITUTION', 'JOURNAL' and 'SUBJECT') is given a list of authority record MARC fields which are to be scanned for data that is to be included in the indexed terms of the dependent bibliographic records. For the 'AUTHOR' authority records, the example specifies that the values of the fields '100__a', '100__d', '100__q', '400__a', '400__d', and '400__q' (i.e. name, alternative names, and year of birth) should all be included in the data to be indexed for any bibliographic records referencing these authority records in their authority-controlled subfields.

Configuration File Overview

The configuration file for the BibAuthority module can be found at *invenio/lib/python/invenio/bibauthority_config.py*. Below is a commented example configuration to show how one would typically configure the parameters for BibAuthority. The details of how this works was already explained in the paragraphs above.

```
# THE FOLLOWING AUTHORITY RECORD TYPES ARE CURRENTLY IN USE
# ... IN THIS INVENIO INSTALLATION:
#
#   'AUTHOR'
#   'INSTITUTION'
#   'JOURNAL'
#   'SUBJECT'
#
# IF YOU WISH TO ADD ANY OTHER TYPES, PLEASE ADD THEM HERE FOR REFERENCE
#
# WARNING: These values shouldn't be changed on a running INVENIO
installation
# ... since the same values are hard coded into the MARC data,
# ... including the 980__a subfields of all authority records
# ... and the $0 subfields of the bibliographic fields under
# ... authority control

# CFG_BIBAUTHORITY_RECORD_CONTROL_NUMBER_FIELD
# the authority record field containing the authority record control number
CFG_BIBAUTHORITY_RECORD_CONTROL_NUMBER_FIELD = '035__a'

# CFG_BIBAUTHORITY_CONTROLLED_FIELDS
# 1. tells us which bibliographic fields are under authority control
# 2. tells us which bibliographic fields refer to which type of
# ... authority record
CFG_BIBAUTHORITY_CONTROLLED_FIELDS = {
    '100__a': 'AUTHOR',
    '100__u': 'INSTITUTION',
    '110__a': 'INSTITUTION',
    '130__u': 'JOURNAL',
    '150__a': 'SUBJECT',
    '700__a': 'AUTHOR',
    '700__u': 'INSTITUTION',
}

# constants for CFG_BIBEDIT_AUTOSUGGEST_TAGS
# CFG_BIBAUTHORITY_AUTOSUGGEST_SORT_ALPHA for alphabetical sorting
# ... of drop-down suggestions
# CFG_BIBAUTHORITY_AUTOSUGGEST_SORT_POPULAR for sorting of drop-down
# ... suggestions according to a popularity ranking
CFG_BIBAUTHORITY_AUTOSUGGEST_SORT_ALPHA = 'alphabetical'
CFG_BIBAUTHORITY_AUTOSUGGEST_SORT_POPULAR = 'by popularity'

# CFG_BIBAUTHORITY_AUTOSUGGEST_OPTIONS
# some additional configuration for auto-suggest drop-down
# 'index' : which index to use for this auto-suggest type
# 'insert_here_field' : which authority record field to use
# ... for insertion into the auto-completed bibedit field
# 'disambiguation_fields': an ordered list of fields to use
# ... in case multiple suggestions have the same 'insert_here_field' values
CFG_BIBAUTHORITY_AUTOSUGGEST_OPTIONS = {
    'AUTHOR': {
        'index': 'author',
        'insert_here_field': '100__a',
        'sort': CFG_BIBAUTHORITY_AUTOSUGGEST_SORT_POPULAR,
        'disambiguation_fields': ['100__d', '270__m'],
    },
    'INSTITUTION':{
        'index': 'institution',
        'insert_here_field': '110__a',
        'sort': CFG_BIBAUTHORITY_AUTOSUGGEST_SORT_ALPHA,
        'disambiguation_fields': ['270__b'],
    },
    'JOURNAL':{
        'index': 'journal',
        'insert_here_field': '130__a',
        'sort': CFG_BIBAUTHORITY_AUTOSUGGEST_SORT_POPULAR,
    },
    'SUBJECT':{
        'index': 'subject',
        'insert_here_field': '150__a',
        'sort': CFG_BIBAUTHORITY_AUTOSUGGEST_SORT_ALPHA,
    },
}

# list of authority record fields to index for each authority record type
# R stands for 'repeatable'
# NR stands for 'non-repeatable'
CFG_BIBAUTHORITY_AUTHORITY_SUBFIELDS_TO_INDEX = {
    'AUTHOR': [
        '100__a', #Personal Name (NR, NR)
        '100__d', #Year of birth or other dates (NR, NR)
        '100__q', #Fuller form of name (NR, NR)
        '400__a', #(See From Tracing) (R, NR)
        '400__d', #(See From Tracing) (R, NR)
        '400__q', #(See From Tracing) (R, NR)
    ],
    'INSTITUTION': [
        '110__a', #(NR, NR)
        '410__a', #(R, NR)
    ],
    'JOURNAL': [
        '130__a', #(NR, NR)
        '130__f', #(NR, NR)
        '130__l', #(NR, NR)
        '430__a', #(R, NR)
    ],
    'SUBJECT': [
        '150__a', #(NR, NR)
        '450__a', #(R, NR)
    ],
}
```

Librarian's User Manual

How to MARC authority records

When adding an authority record to INVENIO, whether by uploading a MARC record manually or by adding a new record in BibEdit, it is important to add two separate '980' fields to the record. The first field will contain the value "AUTHORITY" in the $a subfield. This is to tell INVENIO that this is an authority record. The second '980' field will likewise contain a value in its $a subfield, only this time you must specify what kind of authority record it is. Typically an author authority record would contain the term "AUTHOR", an institution would contain "INSTITUTION" etc. It is important to communicate these exact terms to the INVENIO admin who will configure how INVENIO handles each of these authority record types for the individual INVENIO modules.

Creating collections of authority record

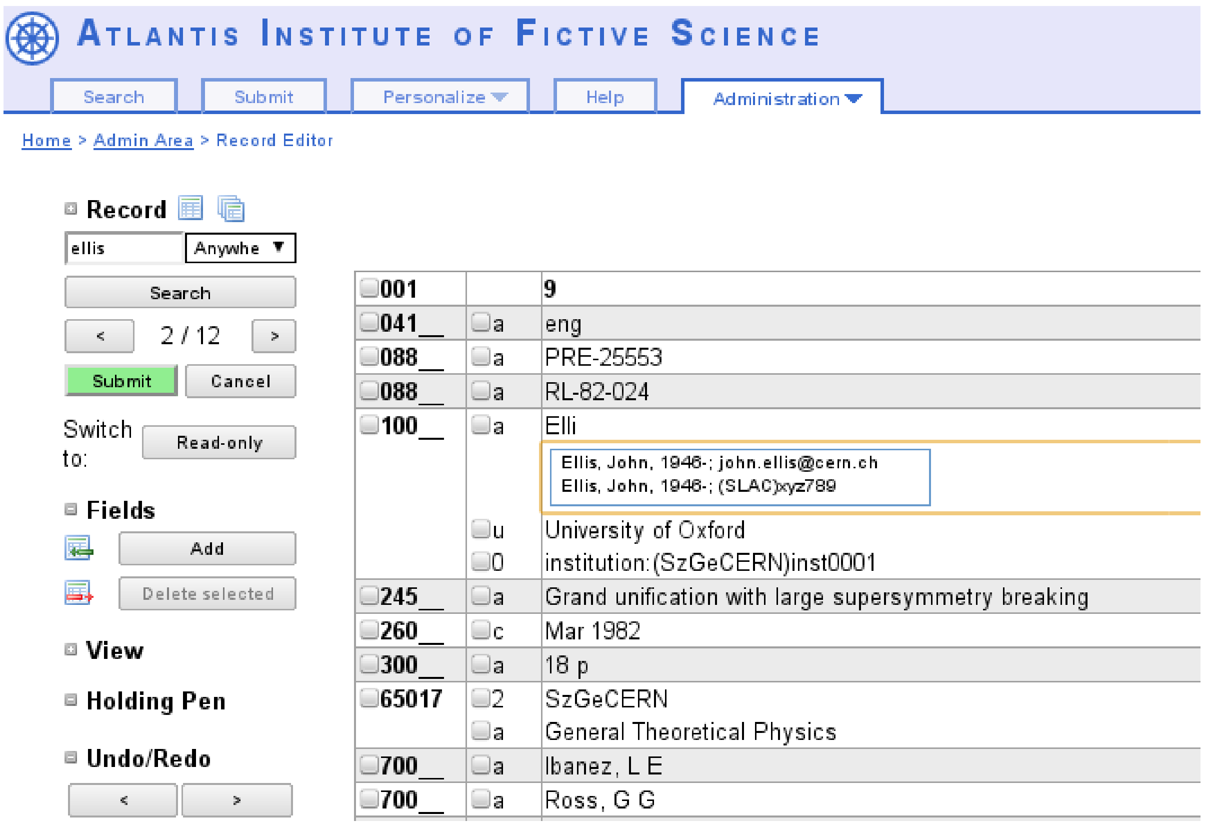
Once the authority records have been given the appropriate '980__a' values (cf. above), creating a collection of authority records is no different from creating any other collection in INVENIO. You can simply define a new collection defined by the usual collection query 'collection:AUTHOR' for author authority records, or 'collection:INSTITUTION' for institutions, etc.

The recommended way of creating collections for authority records is to create a "virtual collection" for the main 'collection:AUTHORITY' collection and then add the individual authority record collections as regular children of this collection. This will allow you to browse and search within authority records without making this the default for all INVENIO searches.

How to use authority control in BibEdit

When using BibEdit to modify MARC meta-data of bibliographic records, certain fields may be configured (by the admin of your INVENIO installation) to offer you auto-complete functionality based upon the data contained in authority records for that field. For example, if MARC subfield 100__ $a was configured to be under authority control, then typing the beginning of a word into this subfield will trigger a drop-down list, offering you a choice of values to choose from. When you click on one of the entries in the drop-down list, this will not only populate the immediate subfield you are editing, but it will also insert a reference into a new $0 subfield of the same MARC field you are editing. This reference tells the system that the author you are referring to is the author as contained in the 'author' authority record with the given authority record control number.

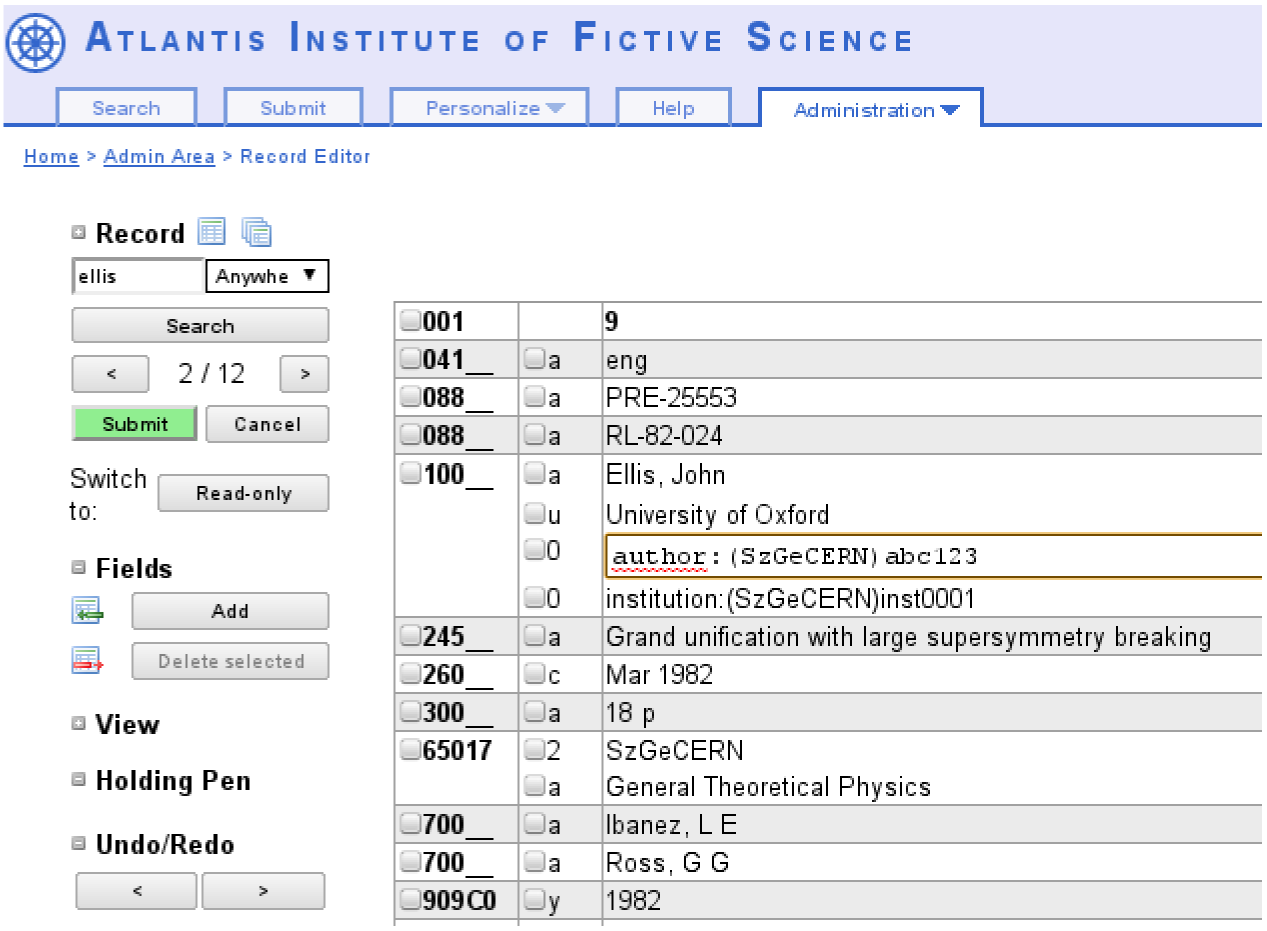
Illustration 1 below demonstrate how this works:



001		9
041__	a	eng
088__	a	PRE-25553
088__	a	RL-82-024
100__	a	Elli
		Ellis, John, 1946-; john.ellis@cern.ch Ellis, John, 1946-; (SLAC)xyz789
	u	University of Oxford
	0	institution:(SzGeCERN)inst0001
245__	a	Grand unification with large supersymmetry breaking
260__	c	Mar 1982
300__	a	18 p
65017	2	SzGeCERN
	a	General Theoretical Physics
700__	a	Ibanez, L E
700__	a	Ross, G G

Illustration 1: Typing "Elli" into the 100__a field triggers an auto-complete dropdown

Typing “Elli” into the 100__ $a subfield will present you with a list of authors that contain a word starting with “Elli” somewhere in their name. In case there are multiple authors with similar or identical names (as is the case in the example shown here), you will receive additional information about these authors to help you disambiguate. The fields to be used for disambiguation can be configured by your INVENIO administrator. If such fields have not been configured, or if they are not sufficient for disambiguation, the authority record control number will be used to assure a unique value for each entry in the drop-down list. In the example above, the first author can be uniquely identified by his email address, whereas for the latter we have only the authority record control number as uniquely identifying characteristic.



001		9
041__	a	eng
088__	a	PRE-25553
088__	a	RL-82-024
100__	a	Ellis, John
	u	University of Oxford
	0	author:(SzGeCERN)abc123
	0	institution:(SzGeCERN)inst0001
245__	a	Grand unification with large supersymmetry breaking
260__	c	Mar 1982
300__	a	18 p
65017	2	SzGeCERN
	a	General Theoretical Physics
700__	a	Ibanez, L E
700__	a	Ross, G G
909C0	y	1982

Illustration 2: The authority record control number is automatically added as well

If in the shown example you click on the first author from the list, this author's name will automatically be inserted into the 100__ $a subfield you were editing, while the authority type and the authority record control number “author:(SzGeCERN)abc123” , is inserted into a new $0 subfield (cf. *Illustration 2*). This new subfield tells INVENIO that “Ellis, John” is associated with the 'author' authority record containing the authority record control number “(SzGeCERN)abc123”. In this example you can also see that the author's affiliation has been entered in the same way as well, using the auto-complete option for the 100__ $u subfield. In this case the author's affiliation is the “University of Oxford”, which is associated in this INVENIO installation with the 'institution' authority record containing the authority record control number “(SzGeCERN)inst0001”.

If INVENIO has no authority record data to match what you type into the authority-controlled subfield, you still have the possibility to enter a value manually.