LHCb Tag Collector

Paloma Fuente Fernández, Marco Clemencic and Nicolas Cousin on behalf of the LHCb Collaboration
CERN, Route de Meyrin, 1211 Geneva 23, Switzerland
E-mail: pfuentef@cern.ch, marco.clemencic@cern.ch and nicolas.cousin@cern.ch

Abstract. The LHCb physics software consists of hundreds of packages, each of which is developed by one or more physicists. When the developers have some code changes that they would like released, they commit them to the version control system, and enter the revision number into a database. These changes have to be integrated into a new release of each of the physics analysis applications. Tests are then performed by a nightly build system, which rebuilds various configurations of the whole software stack and executes a suite of run-time functionality tests. A Tag Collector system has been developed using solid standard technologies to cover both the use cases of developers and integration managers. A simple Web interface, based on an AJAX-like technology, is available. Integration with SVN and Nightly Build System, is possible via a Python API. Data are stored in a relational database with the help of an ORM (Object-Relational Mapping) library.

1. Introduction
The LHCb Tag Collector was born as a system to allow configuring new software releases and Nightly Builds with ease. The aim of this paper is to present this new LHCb Tag Collector.

1.1. LHCb Software
LHCb software consists of hundreds of packages. A set of packages released together is called project. A project version uniquely defines the versions of the packages it contains, and of the projects it depends on.

A package is a group of classes in a logically cohesive physical unit and the minimal entity that can be versioned[1, 2]. Packages have version numbers formatted according to convention: vXrYpZ, with:

- X - Major version: indicates a change in the interface (packages that use it may have to be adapted).
- Y - Minor version: indicates an internal only change.
- Z - Patch version: a minor bug fix to an existing release (optional).

A package can be modified and developed simultaneously by several people. In order to improve the work-flow, a version control system (Subversion[3], also known as SVN) is used.

When a developer introduces changes in a package he maintains, he has to commit them to the Subversion[4] repository and publish the changes on the Tag Collector so that they are included in an official tag and used in the next release. The release manager gets from the Tag Collector the list of packages that require new tags, creates them, then he validates and requests the release.

Published under licence by IOP Publishing Ltd
1.2. Nightly Build System

In order to validate the code and efficiently prepare new releases, a system has been developed to routinely and automatically build the LHCb software projects on the supported platforms (Linux and Windows) and with the supported compilers: the Nightly Build System[5, 6].

The Nightly Build System organizes the builds in slots. A slot consists of a collection of software projects to be built together. Inside a slot, a project can be configured defining the versions of each contained package.

2. Design and implementation

The LHCb Tag Collector is designed loosely after the Model-View-Controller pattern for the main functionality, with extensions to integrate it with the software repository and the Nightly Build System. Figure 1 shows a schema of the implementation components.

2.1. The model

The Tag Collector data are stored in a relational database (Figure 2), currently using the SQLite library[8]. The API to manipulate the model data is written in the Python scripting language[9], using the Object to Relational Mapping (ORM) library Storm[10] for the low level interaction with the database.

We used Python for the API to the model because it is one of the LHCb supported languages, thus giving us the possibility to write tools that interact directly with the model bypassing the
regular, web-based, user interface.

The choice to use an ORM library for the interaction with the database simplifies the implementation, improve the design of the code and allows for a future migration of the relational database from the current SQLite back-end to other back-ends (e.g. Oracle). In particular, we chose the Storm library because of its simple and clean design.

2.2. The view
The user interface (the view component) is implemented as a set of dynamic web pages. The HTML static code is dynamically extended with the content of the model with a modification of AJAX where the asynchronous messages between the HTTP server and the JavaScript code are JSON (JavaScript Object Notation[11]) instead of XML (JSON is lighter and faster to parse than XML).

The JavaScript code, that is the core of the dynamic pages, is written using the jQuery and jQueryUI libraries (very powerful and flexible cross-browser JavaScript libraries[12, 13]) as foundations.

2.3. The controller
The interaction between the view and the model (the controller component) is implemented as a thin layer of CGI scripts written in Python, mainly intended to translate the HTTP messages coming from the JavaScript into Python calls to the API of the model and the Python results into JSON strings to be sent back to the JavaScript.

2.4. Integration with Subversion
We implemented a CGI-based interface to the LHCb software repository using the PySVN python library[14]. All the basic SVN functionalities are available through this interface.

The Tag Collector uses the CGI interface to the software repository to enhance the user experience by proposing auto-completion of the fields from the content of the repository.

It is also possible, for release managers, to create tags for the packages and to view the commits in the repository that the authors have not published in the Tag Collector.
2.5. Integration with the Nightly Build System

The Nightly Build System can be used in conjunction with the Tag Collector to automatically validate the release candidates of the LHCb software projects.

A dedicated CGI script in Python uses the Tag Collector model API to get the list of packages that have to be built in a nightly build slot, with their proposed revisions and tags. The list is then formatted in XML using the same XML Schema that the Nightly Build System uses for its configuration.

The Nightly Build System retrieves the configuration for the build with an HTTP request to the CGI script.

3. Authentication and privileges

It is fundamental for a web application like the Tag Collector to have an authentication system, so that its data are not corrupted by unauthorized users.

The authentication system implemented for the Tag Collector is based on the Single Sign On (SSO) service provided by CERN-IT[15]. This simplifies the implementation and allows us to use the SSO groups to control the level of access to the Tag Collector functionalities.

Three access levels, or roles, have been identified:

- **User**: basic role, read-only access
- **Developer**: limited privileges, allows operations on packages, but not on projects
- **Manager**: unlimited privileges

Three SSO groups are mapped to these roles, with the special case of the User role that is applied also to non-authenticated users.

The CERN SSO service is based on Shibboleth[16] and requires that the pages that need authentication are put in specially configured directories on the web server (Figure 3). To avoid unnecessary load on the authentication system, we put in the special directory only a reduced number of files, caching the session and roles in cookies. In particular we protect the login page
4. User Interface
A menu bar is visible at the top of each web page of the Tag Collector. It contains links to the pages that give access to the available functionalities, a link for the login/logout and some useful links. Depending on the role of the user, a different set of functionalities is displayed, so that basic users do not see the links to the pages that allow changes in the database, while developers are allowed to modify packages and managers have access to all the functionalities.

The main page of the Tag Collector shows different sections, one per group of projects (consistent stack of projects) with a short description of the group. Each group contains one section per project in the group, with the description of the project, the base version, the upcoming version and the list of packages that have been changed since the base version and that are needed in the upcoming one.

Some pages are forms that allow the creation of new entries in the database (e.g. new package revisions or new groups). Other pages give access to advanced views on the database, for example to see which packages contain changes in the repository that have not been published in the Tag Collector.

5. Conclusions and future plans
The LHCb Tag Collector is a young project, but it has already become an essential tool for users, developers and managers, making their work with LHCb software easier.

With the user base still growing, new features are frequently requested. The flexible design of the Tag Collector allows us to easily fulfill the requests.

References
[1] Arnault C 2000 CMT: A software configuration management tool International Conference on Computing in High-Energy Physics and Nuclear Physics (CHEP 2000), Padova, Italy, 7-11 Feb 2000
[2] CMT Home Page URL http://www.cmsite.org/
[3] Apache Subversion URL http://subversion.apache.org/
[4] Clemencic M et al. 2010 Migration of the Gaudi and LHCb software repositories from CVS to Subversion International Conference on Computing in High-Energy Physics and Nuclear Physics (CHEP 2010), Taipei, Taiwan, 18-22 Oct 2010
[5] Kruzelecki K, Roiser S and Degaudenzi H 2010 J. Phys. Conf. Ser. 219 042042
[6] LHCb Nightly Builds – Summary Page URL http://cern.ch/lhcb-nightlies
[7] LHCb Tag Collector – Main Page URL https://cern.ch/lhcb-tag-collector
[8] SQLite URL http://www.sqlite.org/
[9] Python Programming Language Official Website URL http://www.python.org/
[10] Storm Home Page URL https://storm.canonical.com/
[11] JSON, JavaScript Object Notation URL http://www.json.org/
[12] jQuery URL http://jquery.com/
[13] jQueryUI URL http://jqueryui.com/
[14] PySVN URL http://pysvn.tigris.org/
[15] Ormancey E 2008 J. Phys. Conf. Ser. 119 082008
[16] Shibboleth URL http://shibboleth.internet2.edu/