BibTEX++ Plugin system

Youssef Aoun
Supervisor : Ronan Keryell

The 1st March 2005

ENGINEERING PROJECT REPORT

2004-2005

Document version 1.0

ENST Bretagne
Technope Brest-Iroise - CS 83818 - 29238 Brest Cedex 3 - France

Abstract

This project deals with the Plugins aspect of BibTeX++, a bibliography manager for the LaTeX environment.
First of all we will start explaining the context. Afterwards the added functionalities will be exposed and discussed.
It is important to mention that this is a functionality development project. We will not be discussing research matters.

Keywords : BibTeX, compiler, BibTeX++, bibliography, LaTeX, Plugins

Acknowledgments

Firstly, I should address my mega thanks for my supervisor Mr. Ronan KERYELL , the Godfather of BibTEX++, for his valuable ideas and assistance.
I also want to thank Mr. Emmanuel DONIN DE ROSIERE, Mr. Guillaume DUC and Mr. Fabien DAGNAT for there interference.

Contents

Part I

First step

Chapter 1

BibTEX++ newbies' help

1.1 LATEX for dummies

1.1.1 Definitions

- LATEX
  (LAmport TEX) A document preparation system based on the TeX language developed by Leslie Lamport at SRI International. LATEX provides a macro language for TeX that lets the user concentrate on the logical structure of the document rather than the format codes.[4]

- TEX
  A typesetting language developed by Donald Knuth that is noted for its ability to describe elaborate scientific formulas and is widely used for mathematical book publishing throughout the world. Knuth created the first prototype of TeX in 1978 to avoid having to continue using the Unix troff program to typeset his books, "The Art of Computer Programming." It took eight years of writing and revising to complete the language. Available for DOS, Windows, Mac, Unix and other platforms, visit the TeX User Group at www.tug.org for more information.[4]

1.1.2 Necessary tools

To write a basic LATEX document you need the following software:

- The LaTeX compiler and it companion tools[1] .
- Text editor.
  I used TeXMaker[2].

1.1.3 The first LaTeX file

Open your text editor, type these lines and save it as: *text.tex* for example.

```
\begin{document}
\large Hello LaTeX
\begin{equation}
\sqrt{\pi}
\end{equation}
\end{document}
```

1.1.4 How to compile and visualize

1.1.4.1 Step 1: Compilation

The first step consists of a simple compilation using the *LaTeX* command line tool:
latex text.tex You should have a style file such as *plain.bst*. The result is:

- *text.aux*
  Necessary for step 2.
- *text.bbl*
  Necessary for step 2.
- *text.dvi*

1.1.4.2 Step 2: Generating the bibliography

Now you should use either *bibtex* or *bibtex++* to generate the bibliography. For *bibtex* user type *bibtex text*, for *bibtex++* users type *bibtex++ -i text*[3].

[1] http://www.tug.org/teTeX/
[2] http://www.xm1math.net/texmaker/
[3] This document is intended to introduce BibTeX++

1.1.4.3 Step 3: Re-compilation

Now you should re-compile twice in order to concatenate the bibliography and generate the associated numbers. Thus type: *latex text.tex* twice.

1.1.4.4 Step 4: Generating the final output

1.1.4.4.1 Generate XDVI :

Normally at this stage you should have a file called *text.dvi*, type *xdvi text.dvi* to see the result.

1.1.4.4.2 Generate PS :

To generate the associated PostScript file type: *dvips text.dvi -o text.ps*.
Use a common PostScript viewer such as GS.

1.1.4.4.3 Generate PDF :

The precedent step eases to us the PDF creation. Simply type *ps2pdf text.ps* to obtain your PDF file.

1.2 Subjects to learn

BibTEX++ is built to parse file written with a particular syntax. A lexical and syntaxic analysis program has been added to it to generate the parsing object according to grammar. The program is called SableCC[4].
To understand better what is going on, we advise you to check the following subjects:

- Theory of compilers[9][**?**] (Lexical and syntaxic analysis).
- SableCC[8] and it grammar syntax[5].
- Regular expressions[5].

[4]http://www.sablecc.org

Part II

Introduction to the subject

Chapter 2

Introduction

2.1 What is BibTEX?

BibTEX[7] is a program and file format designed by Oren Patashnik and Leslie Lamport in 1985 for the LaTEXdocument preparation system.
The format is entirely character based, so it can be used by any program (although the standard character set for accents is TEX). It is field (tag) based and the BibTEXprogram will ignore unknown fields, so it is expandable. It is probably the most common format for bibliographies on the Internet.

The BibTEXprogram uses style files[2], a list of citations from LATEX, and a BibTEXdatabase[6] to create a LaTEXfile listing the cited references.
Example of a reference:

```
@article{Gettys90,
   author = {Jim Gettys and Phil Karlton and Scott McGregor},
   title = {The {X} Window System, Version 11},
   journal = {Software Practice and Experience},
   volume = {20},
   number = {S2},
   year = {1990},
   abstract = {A technical overview of the X11 functionality.
               This is an update of the X10 TOG paper by Scheifler & Gettys.}
}
```

2.1.1 Format description

2.1.1.1 Special features

This section will discuss briefly the different special features.

The @STRING command is used to define abbreviations for use by BibTeX. The command @stringjgg1 = "Journal of Gnats and Gnus, Series 1" defines 'jgg1' to be the abbreviation for the string "Journal of Gnats and Gnus, Series 1". Any reference outside of quotes or braces to jgg1 will be filled in with the full string.

The @PREAMBLE command is used to define formatter code that will be output directly to the bbl file produced by the BibTeX program. This usually consists of LaTeX macros. It is unclear what one should do with the fields when converting to a format that does not use TeX.

The @COMMENT command lets you put any text inside it. It isn't really necessary, since BibTeX will ignore any text that isn't inside an entry. However, you can not have an @ character outside of an item.

2.1.1.2 Standard entry types

@article
An article from a journal or magazine.
@book
A book with an explicit publisher.
@booklet
A work that is printed and bound, but without a named publisher or sponsoring institution.
@conference
The same as inproceedings.
@inbook
A part of a book, which may be a chapter (or section or whatever) and/or a range of pages.
@incollection
A part of a book having its own title.
@inproceedings
An article in a conference proceedings.
@manual
Technical documentation.
@mastersthesis

A Master's thesis.
@misc
Use this type when nothing else fits.
@phdthesis
A PhD thesis.
@proceedings
The proceedings of a conference.
@techreport
A report published by a school or other institution, usually numbered within a series.
@unpublished
A document having an author and title, but not formally published.

2.1.1.3 Standard fields

For now I'm going to be lazy and give you what Oren Patashnik wrote about the fields. I'll redo this sometime, including references to how each field should be formatted.

address
Usually the address of the publisher or other type of institution. For major publishing houses, van Leunen recommends omitting the information entirely. For small publishers, on the other hand, you can help the reader by giving the complete address.
annote
An annotation. It is not used by the standard bibliography styles, but may be used by others that produce an annotated bibliography.
author
The name(s) of the author(s), in the format described in the LaTeX book.
booktitle
Title of a book, part of which is being cited. See the LaTeX book for how to type titles. For book entries, use the title field instead.
chapter
A chapter (or section or whatever) number.
crossref
The database key of the entry being cross referenced. Any fields that are missing from the current record are inherited from the field being cross referenced.

edition
The edition of a book—for example, "Second". This should be an ordinal, and should have the first letter capitalized, as shown here; the standard styles convert to lower case when necessary.
editor
Name(s) of editor(s), typed as indicated in the LaTeX book. If there is also an author field, then the editor field gives the editor of the book or collection in which the reference appears.
howpublished
How something strange has been published. The first word should be capitalized.
institution
The sponsoring institution of a technical report.
journal
A journal name. Abbreviations are provided for many journals.
key
Used for alphabetizing, cross referencing, and creating a label when the "author" information is missing. This field should not be confused with the key that appears in the cite command and at the beginning of the database entry.
month
The month in which the work was published or, for an unpublished work, in which it was written. You should use the standard three-letter abbreviation, as described in Appendix B.1.3 of the LaTeXbook.
note
Any additional information that can help the reader. The first word should be capitalized.
number
The number of a journal, magazine, technical report, or of a work in a series. An issue of a journal or magazine is usually identified by its volume and number; the organization that issues a technical report usually gives it a number; and sometimes books are given numbers in a named series.
organization
The organization that sponsors a conference or that publishes a manual.
pages
One or more page numbers or range of numbers, such as 42–111 or 7,41,73–97 or 43+ (the '+' in this last example indicates pages following that don't form a simple range). To make it easier to maintain Scribe-compatible databases, the standard styles convert a single dash (as in 7-33) to the double dash used

in TeX to denote number ranges (as in 7–33).
publisher
The publisher's name.
school
The name of the school where a thesis was written.
series
The name of a series or set of books. When citing an entire book, the the title field gives its title and an optional series field gives the name of a series or multi-volume set in which the book is published.
title
The work's title, typed as explained in the LaTeX book.
type
The type of a technical report—for example, "Research Note".
volume
The volume of a journal or multi-volume book.
year
The year of publication or, for an unpublished work, the year it was written. Generally it should consist of four numerals, such as 1984, although the standard styles can handle any year whose last four non punctuation characters are numerals, such as '(about 1984)'.

2.1.2 Common problems

* The original documents specified a large number of field names, but there are many common items that are not listed. A list of some of the ones people have added are below.
* When using BibTeX, the interaction between names and accenting is somewhat tricky. You should use 'Gödel' or 'Gödel', and not 'Gödel' or 'Gödel'.
* The BibTeXprogram is written, as is all TeX, using static data structures, and the maximum length of any one string is by default 1000 characters. It is not uncommon for fields like abstract and contents to overflow this buffer. Solutions to this include
o change the source code to BibTeX (I've changed mine to 3000)
o use include{file.tex} to include an external file
o split the field into field1, field2, ...

2.2 What is BibTEX++?

BibTEX++ [3],is the successor of BibTEX. By conserving the main function of BibTEX, the successor offer more functionalities, better inside design and a multi platform support.

2.2.1 Project's host

BibTEX++ started with an initiative at the department of information technologies at the ENST[1] of Bretagne (http://www.enst-bretagne.fr). The project's website is http://bibtex.enstb.org/

[1]Ecole Nationale Suprieure des Telecommunication - Technopole of Brest Iroise

Chapter 3

Project's development environment

3.1 Java

Java is the development standard for BibTEX++. Concretely in this project Java will be an advantage since the aim is to develop multi platform tool. Besides, dynamic class loading in Java is very developed and could be an advantage for the current project[1].

3.2 External tools and libraries

3.2.1 Apache ANT

Apache Ant is a Java-based build tool. In theory, it is kind of like Make, but without Make's wrinkles[2].
By using ANT, the project's task will be automated [3] . All the newly added features and files has been integrated and compiled using the same philosophy of the previous development.

[1]This project aims to add a plugin call capacity to BibTEX++.
[2]This definition has been taken from http://ant.apache.org/
[3]ANT has not been added during this project. It has been added previously.

3.2.2 Command line parameters analyzer - Getopt

This is a class for parsing command line arguments passed to programs. It is based on the C getopt() functions in glibc 2.0.6 and should parse options in a 100% compatible manner. If it does not, that is a bug. The programmer's interface is also very compatible[4] .
This package has been chosen for its simple and classical use [5] .

3.2.3 Supported operating systems

Since the beginning, the idea was to produce a quality software for multi platforms. The support has been assured for:

- Windows.
- Unix like systems.

[4]The definition has been taken from the Getopt website.
[5]In the next chapters we will argument this choice.

Part III

BibTEX++ - Before and After

Chapter 4

The main strcuture of BibTEX++

4.1 Main object model

Mainly BibTEX++ is composed of four main objects[1] .

- AUX Parser. Used to parse the AUX file created by LaTEX. This object manipulates a set of other objects created by SableCC after a grammar analysis.
- BST Parser. Used to parse the BST style file used by the user. This object manipulates a set of other objects created by SableCC after a grammar analysis.
- BIB Parser. Used to parse the BIB file. This file is the user's bibliography database. The object manipulates a set of other objects created by SableCC after a grammar analysis.
- BibTEX++ main object. It is the main player and the synchronizer of all the objects.

The main object contains the programs sequencing logic in addition to all of the initialization processes.

[1] Check figure 3.1 for more details.

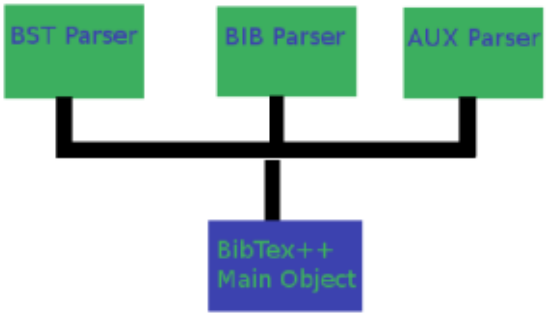

Figure 4.1: The main objects

4.2 The newly added objects

The object model of BibTEX++ permits a relatively easy new features and objects insertion. The new feature, and its relative sub functions, will be as follows:

4.2.1 The plugin management object

The manager is the plugin system's main player. Its duties and efficiency insure BibTEX++ 's internal security and behavior. As main tasks, the manager does the following:

- ➥ Finds plugin JAR files.
- ➥ Checks the files for validation.
- ➥ Opens and loads the JAR's contents for configuration issues.
- ➥ Give an accessibility to the inner parsed objects[2]
- ➥ Makes the plugins available, in different states[3] , for later use.

[2]The BST, AUX and BIB objects, the parsing results[1].
[3]We will be explaining later the plugins' behavior inside BibTEX++.

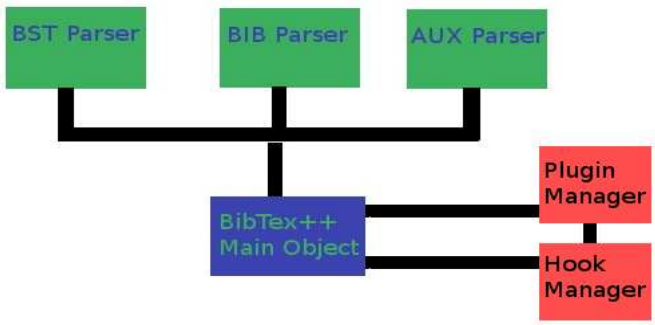

Figure 4.2: The final object model.

4.2.2 The hook management object

4.2.2.1 What is a Hook?

Actually a hook is simply a point in the software on which the desiring plugins are called to work. To be more precise, the plugins are registered to hooks[4] and once this point, or stage, in the program has been reached the registered plugins are called.

4.2.2.2 His work

The hook manager synchronizes the plugins integration within the BibTEX++ internal work.

- ➥ Defines many plugin's interference stages.
- ➥ Receive plugin's registrations for a particular stage of interference.
- ➥ Calls each plugin when is desired stage has been reached.

[4]The hooks who were mentioned in the plugin's manifest file.

Chapter 5

Plugins

In this chapter we will be explaining the constitution of plugins. Their minimal code, exceptions, mode of life..etc.

5.1 Composition and life cycle

5.1.1 BibTexppPlugin - Java Interface

```
public interface BibTexppPlugin
{
        public void setAux(AuxFile aux);
        public void setBib(BibFile bib);
        public void setBst(BstFile bst);
        public void setStage(int stage);
        public void setManagerInterface(ManagerInterface managerinterface);
        public void init(Vector input) throws BibTexppPluginException;
        public void start(Vector input) throws BibTexppPluginException;
        public void destroy(Vector input) throws BibTexppPluginException;
        public String plugInfo() throws BibTexppPluginException;
}
```

For more details on the API you should check the JavaDoc of the project. Whereas in this document we will go over the usage of the different methods encapsulated in this interface.

The plugin should define:

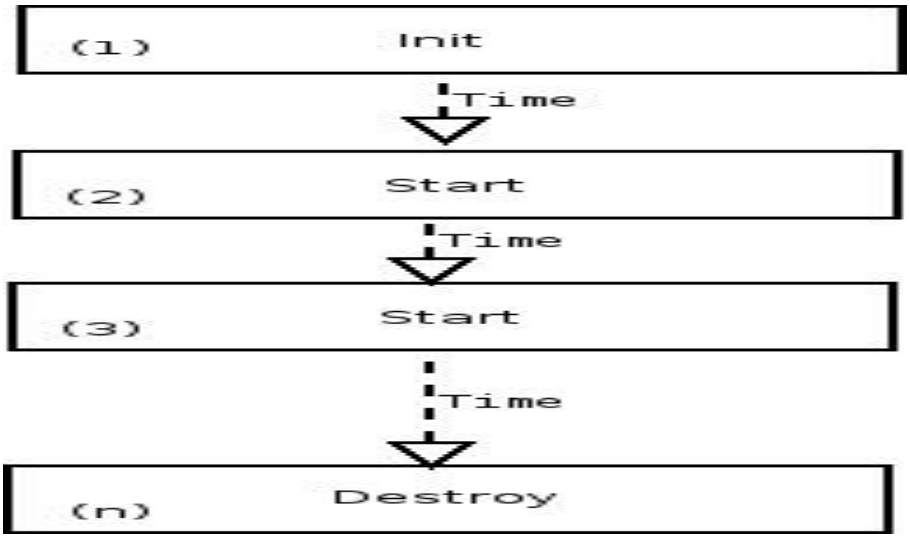

Figure 5.1: Plugins life cycle

- Four setter methods.
  Called by the manager to set the parsed objects and the plugin-¿manager communication object.
- Three action methods.
  Each method is called depending on BibTEX++'s execution stage[1] .
- One information method.
  Called to print information about the plugin.

5.1.2 Life cycle stages

Once inside BibTEX++, the plugin's life cycle could be devised into three parts:

- Instantiated and initialized.
- Running.
- Destroyed

[1]This will be discussed in the next paragraph.

It is important to mention that a plugin's life cycle time and duration depend on what the concepter wants. In fact a plugin could stay always in running mode[2] , pass once to running mode then die or finally, once passed to running mode, stay there until the end[3] .

5.1.2.1 Instantiated and initialized

The plugin is initiated and all its variables are set and he is supposed to communicate his interference preferences[4] . After this phase, it becomes available for solicitation and calling

5.1.2.2 Running

After being loaded, the plugin is stored and ready for calls. This status is called "Running" status. In fact during his initialization phase, the plugin has been registered to the different "Hooks" his manifest file says it should be registered to. During the BibTEX++'s execution and upon the arrival to the selected hook, the plugin's "Start" method will be called after setting an internal variable with the correct value the current hook number.
Example:
If the current hook is the hook number 1, thus the plugin's internal variable "hookstage" will be set to 1 and then the start method is called [5] .

5.1.2.3 Destroyed

At the end of its life cycle, the "Destroyed" method will be called.

5.1.3 Life cycle's behavior, time and options

We distinguish three types of life cycle settings:

- always
  Which will be available and in an active state since the beginning of the excution of BibTEX++. Thus all its internal data will be conserved.

[2]Of course during the BibTEX++ process time.
[3]We will get back to this matter in the following section.
[4]Actually these preferences are written inside the "manifest" file of the JAR compilation of the plugin.
[5]We will discuss this mechanism later in section "Usage".

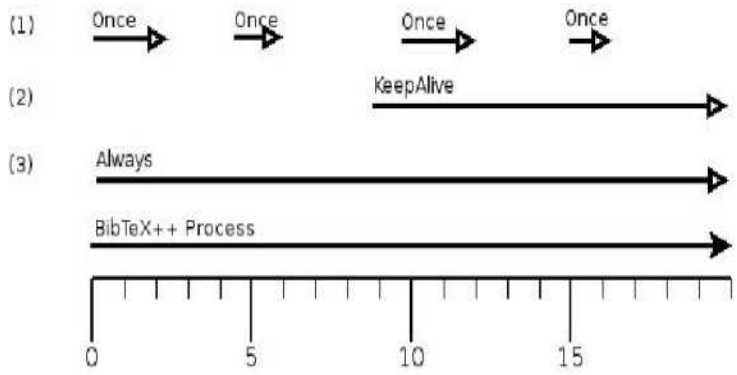

Figure 5.2: Types of cycle behavior

- once
  This plugin will be killed directly after its solicitation but it stays available for another call.

- keepalive
  Once activated, this type will stay alive until the end of BibTEX++'s execution. And since that its activation moment its internal data will be conserved.

5.2 Configuring a plugin

Here is a copy of a manifest file:

```
Manifest-Version: 1.0
Created-By: Youssef Aoun,youssef.aoun@enst-bretagne.fr

Name: BibTexPlugin
PluginName: HelloWorld
PluginBaseClass: helloworld
PluginHook: 1,2,3,4
PluginLife: keepalive
```

For security reasons, all the configuration parameters are stored in the manifest file and are being analyzed by the Plugin Manager. By doing so, the

critical issues will be left for the Manager.
The attributes in the manifest file are essential for the plugin.

- Name
  The name should be always "BibTexPlugin" in order to say that this is a BibTEX++ plugin file.

- PluginName
  This value defines the name by which the plugin will be called later.

- PluginBaseClass
  The following element is very essential. In fact it is the name of the base class of the plugin which implements the BibTex++'s plugin interface.

- PluginHook
  It holds a comma separated serie of numbers each of them defining a hook to register to[6] .

- PluginLife
  This is the famous life cycle type[7] .

[6]The meaning of each number will shown in chapter "The execution time".
[7]It could be : Once, KeepAlive or Always.

Chapter 6

The execution time

6.1 Stages of execution

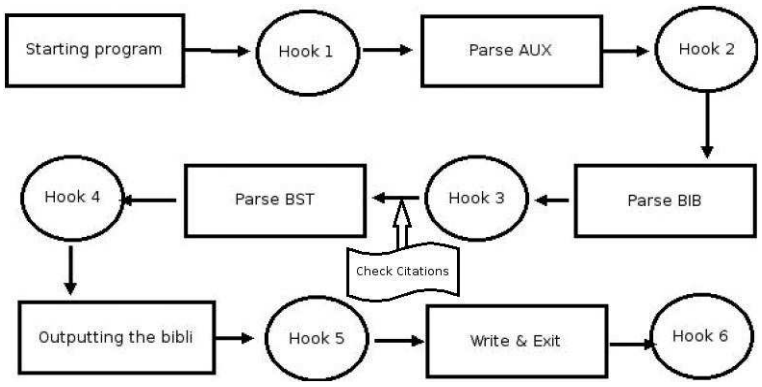

Figure 6.1: Execution diagram

It is important to mention that before the "Starting program" stage, the command line analysis has been done. It is so obvious in fact, since the plugin loading process starts after knowing what do we have to do.
It is obvious that before parsing the BIB file for example, the concerned object is "null". So each plugin, should be enough intelligent to realize this using the "stage" variable.
We identify six important stages in the program:

- Stage 1: Starting program

- Stage 2: Parsing the *.aux* file.
- Stage 3: Parsing the *.bib* file referenced inside the LaTeX document and found in the *.aux* file parsed before.
- Stage 4: Parsing the *.bst* file referenced inside the LaTeX document and found in the *.aux* file parsed before.
- Stage 5: Creating the output
- Stage 6: writing the output and exiting BibTeX++.

There is a sub-stage called "Citations vector scanning" aimed to check if there has been a parameters passing among the citation. This sub-stage occurs just before parsing the *.bst* file.

6.2 Places of hooks

After each important and critical step in the program there is a hook.

- Hook 1 starts after Stage 1.
- Hook 2 starts after Stage 2.
- Hook 3 starts after Stage 3.
  - There is a signal event called "Hook Zero" created by the citation scanner encounters plugin parameters passed among the citations[1]
- Hook 4 starts after Stage 4.
- Hook 5 starts after Stage 5.
- Hook 6 starts after Stage 6.

[1] Check this in chapter "Usage".

Chapter 7

Usage

7.1 Command line parameters

BibTEX++ offers the possibility of command line feeding. Through the shell you may pass different types of parameters aiming to manage the different functions of the program.

7.1.1 Synopsis

```
bibtex++ [OPTION] [PARAMETERS]
```

7.1.2 Description

7.1.2.1 Options

- -v
  Simply prints a textual banner in addition to the software's version and its authors.

- -h
  Prints in a textual mode too a small help text in addition to the banner.

- -n
  If this option is set, normalization will be on.

- -O
  If this option is set, optimization will be on.

- -P
  Charges all the plugins available in the BibTEX++ default plugins repository[1] . Using this option forbids passing special arguments to each plugin throughout the command line.

- -p
  This option, and on the contrary of the previous, it lets you select each plugin you may want to make available and lets you pass parameters to it too.

- -i
  It stands for *"input"*. This option adds a contradiction to the old BibTEXway of work. The input file should be preceded by this option.

7.1.2.2 Parameters

7.1.2.2.1 Options without parameters :

Options who doesn't need parameters are:

- -h
- -n
- -o
- -v
- -P

There use means simple: "Enable function *X*"

7.1.2.2.2 Options with parameters :

- -i
  The only parameter of this option is the name of the *.aux* input file.

- -p
  It is the main option of this project. We may pass through it the plugins' names with parameters to each one of them.

[1] Usually it is BIBTEXHOME/plugins.

7.1.3 Examples

7.1.3.1 Simple and standard use

By typing: *bibtex++ -i report*
You will be invoking the normal and classical BibTEXfunctionalities.

7.1.3.2 Options without parameters

A command that looks like : *bibtex++ [-o] [-n] -i report*

7.1.3.3 Loading all the plugins

bibtex++ -P -i rapport
All the plugins that will be found in the repository will be loaded and made available for later use.

7.1.3.4 Explicit plugin parameters passing through the command line

bibtex++ -p helloworld:High/sam -i rapport
This command will load the plugin called *"helloworld"* and lets the program pass 2 parameters to it in a vector:

- Parameter 1: *High*
- Parameter 2: *sam*

In order to select more than one plugin for loading you may type:
bibtex++ -p helloworld:High/sam -p foo:bar -i rapport
Where *foo* is the plugin and *bar* its one and only parameter.

7.2 Passing parameters inside the LATEX citations

A LATEX citation call looks like:

```
\cite{citationName}
```

Passing parameters inside the citation vector could be a good way to pass plugin parameters specific to each document.
helloworld:high/adam is the syntax to use. BibTEX++ is able to analyze this element and then remove it so it will not be a bug inside the document.
Example:

```
\cite{_PLUGIN_helloworld:high/adam_PARAMS_}
```

By inserting this citation into the *.tex* file you will bi sending parameters:

- high
- adam

for the *helloworld* plugin.

Chapter 8

New ideas

8.1 A grammar for the command line

The actual command line interpreter is *GetOpt*[1] .
Since SableCC is already integrated to our project and that we have already many options and parameters to manage through the command line, we could make a benefit of the lexical and syntaxical analysis by defining a grammar to our command line and creating a more logical command line language.

8.2 Security - Signed plugins

Security, an important issue. The Java Security manager is able to secure the system from the program's behavior. It is worthy to protect the data managed by BibTEX++ from malicious plugins.
The plugin's certification could be a solution. By issuing certificates or by signing the plugins we could at least forbids non signed plugins from executing or track the bugs.
Java[2] offers tools to sign and/or verify the integrety of a plugin.

[1] http://www.urbanophile.com/arenn/hacking/getopt/Package-gnu.getopt.html
[2] Check "jarsigner"

Appendices

Appendix A

Corrected bugs

The concerened source file is:
bst/BuiltIn.java
In the LaTeX source file, the instruction *usepackage[francais]* lets the compiler insert a language name in the ".aux" file. When this instruction does not exist in the LaTeX source file (The writer is assuming that English is the default), nothing will be added in the ".aux" file, thus the parser won't fill the language vector and a java.lang.ArrayIndexOutOfBoundsException exception will be thrown.
Solution:
We added a default return when the execption will be thrown and caught.

```
 private static Collator getCollator(AuxFile aux)
{
   Vector vect=aux.getLang();
   String lang;
   try
   {
    lang=(String) vect.get(0);
   }
   catch(Exception e)
  {
          return java.text.Collator.getInstance(Locale.ENGLISH);
  }
   if(lang.compareTo("french")==0)
      return(java.text.Collator.getInstance(Locale.FRENCH));
```

```
        else if(lang.compareTo("english")==0)
           return(java.text.Collator.getInstance(Locale.ENGLISH));
        else if(lang.compareTo("german")==0)
           return(java.text.Collator.getInstance(Locale.GERMAN));
        else if(lang.compareTo("italian")==0)
           return(java.text.Collator.getInstance(Locale.ITALIAN));
        else if(lang.compareTo("german")==0)
           return(java.text.Collator.getInstance(Locale.GERMAN));

        return java.text.Collator.getInstance();
    }
```

Bibliography

[1] Alfred V. Aho, Ravi Sethi, et Jeffrey D. Ullman. *Compilers, Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA, USA, 1986.

[2] Bibliographix GbR. The Bibliographix website [v?rifi? en Novembre 2002]. `http://www.bibliographix.com`.

[3] Emmanuel Donin de Rosiere. From stack removing in stack-based languages to BibTEX++. Master's thesis, ENST de Bretagne, 2003.

[4] Answers [v?rifi? en March 2005]. `http://www.answers.com`.

[5] Fidel Viegas. Brain creatures [v?rifi? en March 2005]. `http://www24.brinkster.com/araujofh/compiler/sablecc.asp`.

[6] Alain Coulais et Eric Lefvre Anne Buttighoffer. BibTeX pour les nuls [v?rifi? en January 2005]. `http://butti.free.fr/BibTeX.html`.

[7] Marc Baudoin. Apprends LaTeX [v?rifi? en January 2005]. `http://tex.loria.fr/apprends_latex/apprends_latex.html`.

[8] Etienne Gagnon. SableCC, An ObjectOriented Compiler Framework. Master's thesis, McGill University, Montreal, Quebec, March 1998.

[9] Dick Grune. *Compilateurs*. Dunod, DUNOD, Paris, France, 2002.

Powered By BibTeX++©v1.0
Plugin by: Youssef Aoun