Gilbert Babin, John Plaice, and Peter Kropf. CVmaker --- An Activity Report Generator System. Research report no. DIUL-RR-9709. Université Laval. Ste-Foy, Québec, Canada. August 1997.

CVmaker — An Activity Report Generator System

Gilbert Babin, John Plaice and Peter Kropf
{babin,plaice,kropf}@ift.ulaval.ca
Département d'informatique,
Université Laval,
Ste-Foy, Québec, Canada
G1K 7P4

August 1997

Abstract

In most professional organizations, activity reports, summarizing a worker's different functions, must be produced in several formats. This recurrent task is tedious and time-consuming because, although the different documents are produced from the same information, that information is typically stored at different locations in different formats.

The problem of recurrent activity report production can be simplified through the use of a single activity base, which registers all of the various relevant activities, and the use of a structure-oriented language for producing documents from the base. The structure of the activity base must be easy to use and flexible, allowing for new classes of activities and characteristics, while being independent from any database. Furthermore, reports should be produceable for any well-specified output format.

Specifically, the solution we propose uses (1) a BiBTEX-like syntax to define the *Activity Base* and (2) a new structure-oriented language (*CVmaker*) for the definition of the document structure. BiBTEX, a bibliography generation tool used mostly with LATEX, allows for the description of bibliographical references. We have used its basic structure to store activity information.

The *CVmaker Document Description Language* allows the user to define document templates; namely, it defines (1) the sections, sub-sections, etc. of the document; (2) the activity classes included in the different sections; (3) the sorting order of elements within a section; and (4) the output format of the document.

This paper presents the *CVmaker* system in detail: the use of BiBTEXto create the Activity Base, the *CVmaker* Document Description Language syntax, and the *CVmaker* Interpreter prototype. The presentation concludes with the presentation of future work.

1 Introduction

Professional workers in large organizations or academic institutions are often required to produce activity reports, detailed resumes, and other documents describing their various accomplishments. The information found in each document is basically the same, but the format can vary extensively from one document to another. Some of these documents are in paper format, while others are in electronic form, to be input by various commonly used text processing programs.

The task of rearranging the information in the different formats is time consuming and error prone. Not only must the various document formats be mastered (sometimes a complex task in itself), but updates force one to face the problem of simultaneously maintaining many different versions of the same information, clearly poor practice.

This task is well suited to automation. The ideal solution is to use a single database of the activities for an individual, henceforth called the *Activity Base*. The Activity Base should include detailed information about the activities performed by that individual. For such a base to be useful, it must be fully extensible, in the sense that adding new kinds of activity in no way invalidates previous documents. Similarly, the

information in such a base should be sufficiently flexible for the printing of documents in several natural languages.

Once this Activity Base is created, means must be provided to build documents of various forms from the base. We assume that a paper document can be generated from an appropriate electronic document, using appropriate software. Building these documents means one must define (1) their structure; (2) the sort keys required to order the entries in a document; and (3) the output format.

To effect these tasks, there are two possible approaches. The first would be to use some general language such as SGML for the definition of both the Activity Base and the printing of documents. This approach is probably the most general, but SGML authoring tools are not widely available, at least for the moment.

The other approach is to use a widely available and widely used tool, and to extend it as required. This was the choice we made, using BiBTEX, first developed to produce reference lists for LATEX [1, 2], and now the *de facto* standard for electronic bibliographies in computer science, as well as in other disciplines.

The BiBTEX tool has three components: a reference base structure, a postfix notation scripting language to produce reference lists, and an interpreter for the language. There are many advantages in using BiBTEX. First, and foremost, the reference base of BiBTEX is widely used by researchers, especially in scientific research domains, in conjunction with LATEX, to produce research documents. Second, a large number of predefined BiBTEX scripts, called styles, are available, making the task of producing a reference list even easier.

In our first efforts to automate the production of activity reports, we extended the reference base to include research and teaching activities, since we work in an academic environment. The reference base structure is ASCII-based and can easily be expanded. This approach looked promising, in the sense that we could use the same standard format to produce (1) reference lists in research documents and (2) activity reports.

Then, using the BiBTEX scripting language, we programmed style files generating the final documents. This task proved difficult: the postfix notation makes it difficult to follow the logic of a script, complex sort keys must be generated by hand, and we would quickly reach the internal limits of the BiBTEX interpreter. As a result, we decided to no longer use the BiBTEX scripting language.

The *CVmaker* approach was developed to alleviate these problems. *CVmaker* consists of three components: the Activity Base, the *CVmaker* Document Description Language (CDDL), and the *CVmaker* Interpreter. The Activity Base retains the BiBTEX reference base structure. The CDDL was developed to produce structured documents, based on the information found in the Activity Base. The *CVmaker* Interpreter will produce a document from (1) a CDDL script and (2) a series of Activity Base files.

This paper presents in detail the *CVmaker* approach for the repetitive production of activity reports in different formats. In Section 2, we describe the Activity Base, including a presentation of the BiBTEX reference base structure and how it can be expanded for use with *CVmaker*. Section 3 follows with a description of the syntax of the CDDL. The current *CVmaker* Interpreter prototype is presented in Section 4. Concluding remarks (Section 5) include extensions to the *CVmaker* approach.

2 The Activity Base Structure

The Activity Base consists of entries written in the style of BiBTEX files. A BiBTEX file (usually with a `.bib` extension) is composed of a series of BiBTEX entries, whose structure is described in detail in [1, 2]. Basically, each entry contains a reference class identifier, a unique reference key, and a series of field entries. In Figure 1, the reference class is `book`, whereas the reference key is `Goossens94`. Field values are placed between braces or double quotes.

The Activity Base syntax was designed with extensibility in mind. Using the BiBTEX syntax, new classes and fields can be arbitrarily defined to describe activities. The meaning of the classes and fields is established by the users of *CVmaker*. However, two macros are predefined: `t`, meaning "true", and `f`, meaning "false". They are used in the internal processing of *CVmaker* to represent the possible values of Boolean expressions.

For the purpose of this research (and for our personal use), we have defined new classes corresponding to academic and research activities, and new fields, qualifying the activities in old and new classes. For example, Figure 2 presents the relationship between a postdoctoral fellow and her advisors.

```
@book { Goossens94,
        author    = {Goossens, Michel and Mittelbach, Frank and Samarin, Alexander},
        title     = {The \LaTeX\ Companion},
        publisher = {Addison-Wesley},
        address   = {Reading, MA, USA},
        year      = 1994
      }
```

Figure 1: A BiBTEX entry example

```
@advisor { PostDoc:Dupuis:95,
           author      = {Dupuis, Renée},
           editor      = {Toupin, A. and Maltais, R.},
           institution = {Université Laval},
           address     = {Québec, Canada},
           type        = postdoc,
           year        = 1995
         }
```

Figure 2: An Activity Base entry example

3 The *CVmaker* Document Description Language

The *CVmaker* Document Description Language defines (1) the structure of documents to be produced, i.e., the sections composing each document, headers, trailers, etc.; (2) the relationship between entries found in the Activity Base and sections of a document; (3) the sort key for each individual section; and (4) the output format for each activity class appearing in a section.

To illustrate the use of CDDL, we provide small examples of its usage in the subsections below. Note that identifiers are classified as follows: (1) activity classes begin with an '@', (2) fields begin with a '.', (3) variables and routines begin with a '\', (4) reserved CDDL words begin with a '$', and (5) macros begin with a letter.

3.1 Document structure definition

The main purpose of CDDL is to define templates describing the overall structure of documents. (Templates are stored as files with `.tpl` extension). Figure 3 illustrates the main parts of a document template: (1) global parameters and declarations (sections `$parameters` and `$declarations`), which specify language settings, generic output functions, etc.; (2) the document's header and trailer (sections `$header` and `$trailer`); and (3) a list of sections.

The list of sections includes an operator indicating how to generate section numbering: `$alpha`, `$roman`, `$arabic`, and `$none`. The `$none` operator is used when the user wishes to put section numbers directly into section titles. The list is composed of a series of section declarations, each one beginning with the `$section` operator. A section, in turn, can contain a series of entries (`$items` operator) and/or a list of subsections, recursively. In Figure 3, we have two top-level sections, numbered by letters. Furthermore, Section A has n subsections, each of which is numbered by Arabic numerals.

3.2 Activity–structure relationships

The mapping of activities into sections is done within the `$items` clause. This clause describes how the activities composing a specific section are to be selected, processed, and formatted. The selection done as in Figure 4: for each activity class to be selected, we write a `$case` statement, identifying the activity class and providing a selection predicate. In the example, the section "Refereed Publications" contains all refereed journal and conference papers published in the year `\current`, as specified by the two `$case` statements and their associated predicates (`.hasreferee` equals `t` and `.year` equals `\current`). Note that variable `\current` is a global variable, declared by the user, containing the current year.

```
$parameters { ...  }
$declarations { ...  }
$header { ...  }
$alpha {
   $section "Title Section A" {
      $items ...
      $arabic {
         $section "Subsection A.1" { ...  }
         $section "Subsection A.2" { ...  }
         ...
         $section "Subsection A.n" { ...  }
      }
   }
   $section "Title Section B" { ...  }
}
$trailer { ...  }
```

Figure 3: Global structure of a document

```
$section "Refereed Publications" {
   $declarations { ...  }
   $items
      $header { ...  }
      $case @article :  .hasreferee ^ (.year = \current)
         $format { ...  }
      $case @inproceedings :  .hasreferee ^ (.year = \current)
         $format { ...  }
      $trailer { ...  }
   $sort { ...  }
   }
```

Figure 4: Linking activities to a section

3.3 Sort key definition

For each section defined in the document template, we can provide an explicit sort key. Note that in the absence of a `$sort` clause, the sort key is the section order number and the activity entry order number. In fact, all the keys created start with the section order number, to ensure that each entry is processed in sequence, and end with the activity entry order number, to resolve any conflict in sort keys.

We define a sort key by declaring the fields composing it, the order in which to consider those fields, and the relative order of values within that field. Figure 5 shows the use of the `$sort` clause in a section producing a list of books. As mentioned earlier, the section order number is the first element in the key. The next element is the author's name; here, we specify that the sort key is the concatenation of all the authors of a book; for each author, we will consider the subfields in the sequence [`.von`, `.last`, `.jr`, `.first`]. The next element is the last name of the editors of the book. The publication year follows, in reverse order, as specified by the "`(>)`" operator. We then use the book title. Finally, we will place books before book parts.

3.4 Output format description

The production of the actual output is done for each section and for each `$case` clause. At the section level, we can declare local variables, which respect the scoping of nested sections. In Figure 6, we declare the integer variable `\count`, which is used to count the number of postdoctoral fellows being supervised. The section's `$header` clause defines the header output (if any) for the section. In our example, we initialize variable `\count` to 0. The section's `$trailer` clause defines the trailer output (if any) for the section. Here, we print the total number of postdoctoral fellows.

For each entry matched in the `$case` clauses, we define the output format in the `$format` clause. Operations include procedure calls (`$do` statement), conditional statements (`$if` statement), for-loops (`$for`

```
$section "Books" {
   $declarations { ...  }
   $items
      $header { ...  }
      $case @book:  (.type = technical) ^ (.year = \current)
         $format { ...  }
      $case @incollection:  (.year = \current)
         $format { ...  }
      $trailer { ...  }
   $sort {
      .author[.von,.last,.jr,.first];
      .editor.last;
      .year (>);
      .title;
      $classlist ( @book, @incollection );
      }
   }
```

Figure 5: Defining sort keys

```
$section "Postdoctoral Fellows" {
   $declarations { $integer \count; }
   $items
      $header { \count := 0; $newline;}
      $case @advisor:  .type = postdoc
         $format {
            $do \print_authors() ;
            $if .editor <> $empty {
               ", supervisor(s):  ";
               $do \print_editors() ;
            }
         $newline ; \count := \count + 1;
         }
      $trailer { $newline; " Number of fellows:  " ; \count; $newline ; }
   $sort { ...  }
   }
```

Figure 6: Formatting the output

statement, not illustrated in Figure 6), variable assignments (using the ":=" operator), and print statements.

Printing is accomplished in two ways: explicitly, by using the `$print` statement followed by an expression, or implicitly, by writing a string, macro, variable, or field name alone in a statement. Newlines are produced explicitly using the `$newline` statement.

4 The *CVmaker* Interpreter Prototype

A report is produced by the *CVmaker* Interpreter, which uses a document template (`.tpl` file) and a series of Activity Base files (`.bib` files) as input. Processing consists of seven steps: (1) initializing the environment (standard macros and fields definitions); (2) loading the document template (which may override standard definitions); (3) loading the Activity Base; (4) mapping the activities into the document's sections; (5) building the sort keys; (6) performing a heap sort of all the entries; and (7) executing the document template to produce the final document. Figure 7 illustrates the result from the partial document template shown in Figure 6.

The syntax of the document template and that of Activity Base entries are checked while the files are being loaded.

In order to map the activities to the sections, we keep a table of the `$case` clauses involving the different

```
Postdoctoral Fellows

Dupont, M., supervisor(s):  Maltais, R.
Dupuis, R., supervisor(s):  Toupin, A. and Maltais, R.

Number of fellows:  2
```

Figure 7: Final result

activity classes. This reduces the search time. For each activity, we identify the first clause (they are kept in the relative order of the sections) for which the selection predicate is true and assign the activity to the corresponding section. At the same time, we calculate the largest field sizes, in order to build the sort keys. In a second pass, we generate the sort key for each entry. We need two passes, since we need to know the largest possible size for the different fields composing the sort keys. The sort key is the concatenation of the value of all the fields composing the sort key, as explained in Section 3.3; each value is padded with NUL's. Sorting in reverse order is achieved by creating the complement string.

Once the key is built, a heap sort procedure will order the activities. Note that activities not assigned to a section will have an empty sort key, and will therefore be at the start of the sorted list. The processing of the document template will skip these entries. The document is produced sequentially, processing the sorted activities as their corresponding section is reached.

5 Discussion

In the previous sections, we introduced *CVmaker* as a solution to the problem of multiple activity report production. *CVmaker* is regularly used for this purpose by the three authors, and is also being used experimentally by some of our colleagues. From the same activity base, reports have been produced in unformatted ASCII, as well as in LaTeX, HTML and Microsoft Word™'s RTF.

The various *CVmaker* tools are implemented using a home-grown LL(1) compiler generator called CompTools (2400 lines of source), along with an additional 3000 lines of C. The tools run on several Unix™ environments, and would require minimal modification to run in other environments. As for the Activity Base, in addition to the 13 predefined BibTeX classes, we use another 20 classes for our purposes. On average, defining the output for a given class–document combination requires about 20 lines of CDDL.

The results we have obtained to date are sufficient for basic use. Nevertheless, before these tools can be released, certain improvements will have to be made. These improvements, currently being implemented, can be classified in two categories: (1) additions that ensure that the languages and tools being used more resemble standard programming languages and environments (variables, structuring of programs, graphical interfaces, etc.); and (2) additional functionality that is specific to the problem at hand (complex sort keys, multiple languages, etc.) In the following discussion, we will focus on the latter.

As the world economic system becomes more integrated, it is important that programs manipulating electronic documents not simply be designed for English. Within our own institution, we must generate, from the same base, documents in English and in French, using the appropriate conventions for each language (e.g., using « » rather than " " in French). When we interact with other institutions in other countries, we must take into account their conventions as well.

Multiple languages arise in two different contexts. First, the activity base itself may have raw information that is input in several languages. For example, titles of talks given or of articles written may be in several different languages, and this information must be retained, to ensure proper printing of the text. Second, a document normally has a major language or, in the case of a multilingual document, several major languages.

To address the first situation, we are experimenting with various forms of tags to specify that a given string is in, say, English, French or Arabic. By doing so, the appropriate fonts, typesetting routines, etc. can be loaded automatically by the output routines when generating documents.

As for the second situation, it creates two kinds of problems. The first and easiest requires printing headers in the appropriate language(s). The second is more complex: it consists of manipulating multiple sort keys, according to the major language of a document. This problem becomes even more complex in a

multilingual document, where one might want to have different parts of a document ordered using different languages.

Rather than develop specialized routines for each language, we are developing a general means for sorting, in which complex character ordering, usable for any language, will be specifiable.

We conclude by reiterating that this approach to multiple format document preparation has allowed us to generate a series of portable tools that are fully compatible with BibTEX, the widely available standard for electronic bibliographies. We suspect that the utility of these tools goes well beyond their current use.

References

[1] Michel Goosens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion.* Addison-Wesley, Reading, MA, USA, 1994.

[2] Leslie Lamport. *LaTeX: A Document Preparation System — User's Guide and Reference Manual.* Addison-Wesley, Reading, MA, USA, 2nd edition, 1995.