KielceRB: A highly customizable templating engine for course documents*

Zachary Kurmas
School of Computing
Grand Valley State University
Allendale, MI 49401
kurmasz@gvsu.edu

Abstract

We introduce KielceRB: a customizable templating engine for generating assignments, syllabi, web pages, and other course documents. KielceRB loads a hierarchy of key-value pairs from files at various file system levels. These values can then be inserted into web pages and other documents using Ruby's ERB templating engine. KielceRB simplifies the maintenance of course documents by moving data that changes from semester to semester into external data files where they can be easily identified and updated. By loading data from various file system levels, it is easy to share values among all documents for a particular course and/or semester. Data "values" can also be functions, which allow the values appearing in the documents to be composed and/or calculated.

1 Introduction

We introduce KielceRB, a highly customizable templating engine for assignments, syllabi, web pages, and other classroom documents. KielceRB loads a hierarchical set of key-value pairs from a series of data files, then inserts those values into course documents using Ruby's ERB templating engine. Placing

frequently changing items like due dates and software version numbers in a separate data file makes updating a document from semester to semester easier and less error prone.

`KielceRB` also includes a simple deploy system for those instructors who prefer posting course documents on a traditional web site rather than using a Learning Management System (LMS) like Blackboard, Canvas, or Moodle.

1.1 Motivation

We designed `KielceRB` to address several inefficiencies in maintaining course documents. First, maintaining course documents can be error prone. Every semester, the instructor must search through each document and update data like due dates, semester names, and software versions. In our experience, it is easy to overlook updates – especially when a document has several dates (e.g., a syllabus, or an assignment with several deliverables). Moving these frequently-changing values into a separate document makes it easy to identify and update them all.

Second, making the updates can be tedious. Some values, such as the semester name, appear in every document. Similarly, there are values such as office hours and the withdraw deadline that appear in each course's syllabus. Moving these values to a separate document means they need only be updated once per semester, as opposed to once per semester per course.

We also designed `KielceRB` to simplify the process of pushing course documents to a web page. We have noticed that as the use of Learning Management Systems (such as Blackboard, Canvas, and Moodle) has increased over the years, it has become more difficult to casually browse the web and see how other instructors are approaching their courses. We believe this inhibits innovation; therefore, we make a point of posting our course materials where they can be easily found by other instructors and indexed by search engines. `KielceRB` supports this goal by providing scripts to (1) separate public content (e.g., the assignment description and any "starter code") from private content (e.g., solutions and notes) and (2) push that public content to a web server.

Finally, our design emphasizes *customizeability* over *configurability*. Many frameworks and tools make a point of being configurable: They offer many settings to allow users to adjust the tool to their own preferences. The challenge (as anybody who has spent time customizing a new IDE knows) is that it often takes a lot of time and effort to learn about all the options and how to use them. In the field of Computing education, many instructors choose to write their own tools and assignments rather than either (1) conform to the patterns established in an existing tool, or (2) take the time to learn how to effectively use all the configuration options.

Rather than providing a long list of configuration options that few instructors will take the time to learn, we decided to make `KielceRB` as simple as possible and encourage instructors to simply *customize* (i.e., re-write) it as they see fit. This approach certainly doesn't work for educators as a whole; but, we believe it is a good match for how many Computing instructors prefer to work (especially those who like to code).

2 Templating Engine

At a high level, `KielceRB` loads a set of key-value pairs from a set of files, then uses Ruby's ERB templating engine to insert those values into a document. A data file can be as simple as

```
{
   name: 'Tom'
}
```

The most straightforward way to insert that value is to add ERB's `<%= %>` tags to a document: (`KielceRB` loads the data file into a global object named `$d`.)

```
  Hello, <%= $d.name %>. How are you today?
```

Generating the final product is as simple as running

```
kielce.rb inputFile.erb > outputFile.html
```

ERB is designed for the generation of HTML documents; but instructors can just as easily use ERB and/or `KielceRB` to help prepare any text document (e.g., LaTeX).

In addition to basic templating, `KielceRB` provides a few additional features to support its use for maintaining course documents:

2.1 Hierarchical data object

A `KielceRB` data file is Ruby code that returns a hash. As Figure 1 shows, the values of this hash can themselves be other hashes, allowing users to organize data hierarchically. This hash is converted into an object with properties instead of key/value pairs so that data can then be inserted into documents using the more convenient method call syntax:

```
Drop deadline: <%= $d.semester.dropDeadline %>
```

```
{
  course: {
    prefix: 'CIS',
    number: '371',
    name: 'Web Application Programming',
  },

  semester: {
    year: 2020,
    term: 'Winter',
    dropDeadline: 'Friday, 6 March',
  },

  general: {
    myURL: 'https://www.myfavorite.edu/~smith'",
    honestyURL: 'https://www.myfavorite.edu/academic-honesty',
  },
}
```

Figure 1: Sample `KielceRB` input file

```
Courses
 + WebProgramming
   + Homework
     + WebServer
     + WebBrowser
     + CSS
 + ComputerArchitecture
   + Homework
     + Processes
     + Threads
     + Scheduling
```

Figure 2: Sample file structure for course documents

```
general: {
  piazza: lambda do |data|
   "https://piazza.com/gvsu/#{data.root.sem.piazzaName}/#{data.root.course.id}"
  end
},

course: {
  prefix 'CIS',
  id: -> (data) { "#{data.local.prefix}#{data.local.number}" },
},

sem: {
  year: 2020,
  term: 'Winter',
  dropDeadline: 'Friday, 6 March',

  fullName: -> (data) { "#{data.local.term} #{data.local.year}" },
  shortName: -> (data) { "#{data.local.term[0, 1]}#{data.local.year % 100}" },
  piazzaName: -> (data) { "#{data.local.term.downcase}#{data.local.year}" }
}
```

Figure 3: Semester data for moderately complex configuration

2.2 Hierarchy of data files

To facilitate the sharing of data among several courses and/or assignments, `KielceRB` merges the hashes from several files into a single hierarchy. Specifically, the tool will search all ancestor directories for files that begin with `kielce_data` and end with `.rb`. Thus, when using a typical file structure like the one shown in Figure 2, data shared among all course documents can be placed in `Courses/kielce_data_general.rb` (for example, items in the `general` and `semester` groups above); data shared among all documents for a specific course can go in a data file in that course's directory (e.g., `Courses/WebProgramming/kielce_data_wp.rb` and `Courses/ComputerArchitecture/kielce_data_ca.rb`); and, data for each specific assignment can go directly in that assignment's directory. (It is not necessary for each data file to have a unique name. We do that because it is easier to see at a glance which file we are currently editing when we are working on multiple files at once.)

2.3 Functions

The values can also be functions (i.e., Ruby lambdas), allowing us to write code to compute and/or compose values. For example, we use a `semester` data

```
course:  {
  number: 371,
  refDir: 'Manuals',
  devManual: -> (data) {"#{data.local.refDir}/IntelDev_v#{data.args[0]}.pdf"},
  }
```

Figure 4: Data for one course

section similar to that shown in Figure 3. The `fullName` lambda combines the `term` and `year` values into the single string "Winter 2020" — allowing us to reference the "full name" of the semester as a single entity. The `shortName` lambda automatically generates the abbreviated string "W20". Thus, each semester, we need only update `year`, `term`, and `dropDeadline`; the other values get updated automatically.

Notice also that our semester data file also contains some course data – namely `prefix` and the `id` lambda. This is because the prefix is the same for all of our courses. Similarly, the rule for creating the course id (e.g., "CIS371") is identical for all courses. Therefore, we place these items in a file that is used by all courses. Similarly, the `piazza` lambda in the `general` section is also identical for all courses. Both lambdas rely on data that is unique for each course. In other words, we can establish "high-level" rules, but allow the specific data to be provided at a "lower" level.

The `data` parameter to the lambdas contains three objects:

- `root` refers to the root of the data hierarchy. Thus, the `fullName` lambda could have alternatively accessed the year as `data.root.semester.year`.
- `local` refers to the node in the data hierarchy that lambda is declared in. For example, in the `fullName` lambda, `data.local` refers to the `semester` block. (This option serves primarily as a shortcut for users who employ a deep hierarchy.)
- `args` refers to arguments passed to the lambda. Figure 4, demonstrates how users can generate the URL for a local copy of Volume 3 of the Intel Developer's Manual using syntax like `<%= $d.course.devManual(3) %>`. The actual parameter `3` is made available in the lambda as `data.args[0]`.

The examples above are all relatively simple, one-line functions; but, functions can be as long and complex as the user desires. For example, we have used longer functions to (1) format the course textbook (placing the title in italics, creating a link to the book's web site, etc.), and (2) generate a list of a courses's weekly labs.

2.4 Plug-ins

`KielceRB` also contains a rudimentary plug-in system, where users can place code that is too long and/or complex to reasonably be contained in a well-organized, maintainable data file. Specifically, `KielceRB` looks for directories named `kielce_modules` and loads any Ruby modules found there. Those modules are then made available to the lambdas in the `kielce_data` files.

To demonstrate the usefulness of plug-ins, we wrote a plug-in that generates a timeline for a course based on data in an Excel spreadsheet. The plug-in code uses the `rubyXL` Ruby gem to read the Excel file, then generates two HTML tables: One large table containing a day-by-day listing of course topics, readings, references, and due dates. The second, smaller table just lists assignment deadlines and is placed on the course's main page. Web pages generated using this plug in can be seen here: `https://bit.ly/2YlolEF`

2.5 Miscellaneous

Finally, we mention a few minor, but helpful, features:

- `KielceRB` can import one file into another. This makes it easy, for example, to include a common CSS file on all pages, or add a common navigation bar to all pages.

- The tool adds a `link` method to the `String` class, which allows users to quickly generate a link from a string:

  ```
  <%= 'https://somereference.com/topic'.link() %>
  ```

  When we include a link in a course assignment, we often want the text of the link to be the URL itself so students can easily see and copy it, if desired. This short-cut allows us to avoid typing the URL twice.

3 Deploy Helper

We prefer to deploy assignments and other course documents to our university-provided web pages rather than an LMS. First, our university's LMS offers neither an API nor a command-line interface, so we must manually navigate through a slow, awkward, inefficient web iterface to deploy documents. As a result, an operation that should take less than a second takes tens of seconds. Second, we prefer to share our content with the educational community to help propagate new and innovative assignments and approaches to teaching. Although most LMS content can be made accessible to all, in our experience, traditional web pages are easier to search and navigate.

To facilitate the deployment of course documents, `KielceRB` includes a deploy helper that automatically copies documents into a local mirror of the

```
function prepare() {
  kielce.rb -q css.html.erb > $target/index.html

  cp Images/* ${target}/Images
}
```

Figure 5: Typical `prepare_assignment` script

teaching section of our web site. We then use `rsync` to push those changes to the actual web page.

The deploy helper uses the "convention over configuration" principle [5]. It assumes that our course documents are organized in a structure similar to that shown in Figure 2 (specifically, a `Courses` directory containing a subdirectory for each course). The teaching section of the web site is also assumed to maintain a parallel structure.

When run, the deploy helper does the following:

1. Creates a directory in the local mirror, if necessary. For example, if the helper is run from within `Courses/WebProgramming/Homework/CSS`, then a directory with the same name is created in the local mirror;

2. Runs a bash script named `prepare_assignment` that prepares the public documents and places them in the target directory within the local mirror; and

3. Creates a sym-link to the target directory.

Figure 5 shows a typical `prepare_assignment` script. This script must contain a function named `prepare`. The helper initializes `$target` to the target directory before calling `prepare`. In this example, the first step is to run `KielceRB` and redirect the output to the target directory. Next, other supporting documents (e.g., images) are copied to the target directory. The helper copies an explicit list of files to a separate target directory (as opposed to simply creating a sym-link to the current directory) because many assignment directories contain solutions or other files that are not intended to be public.

4 Customizable vs Configurable

Many complex software products, such as IDEs are *configurable*: There is a large configuration file with many options that allows the user to specify desired behavior. `KielceRB` takes a different approach to flexibility: The source code

is sufficiently simple that the best way to make changes is to simply edit (i.e., "*customize*") it. The core code is only 350 lines long including comments. Adding a configuration system would make this code more difficult to follow and limit the types of changes that can easily be made.

The most complex parts of the templating system are:

- Merging the various `kielce_data` files, and
- Converting the resulting merged hash into an object.

We don't anticipate much demand for customization in this area. We do expect that different users may prefer different conventions for naming and finding the data files. In which case, the most flexible way to support that change is to encourage the users to simply replace those few lines of code. For example, `KielceRB` currently runs this line of code at each level of the directory hierarchy to find input files:

```
data = Dir.glob("#{dir}/KielceData/kielce_data*.rb") +
       Dir.glob("#{dir}/kielce_data*.rb")
```

We believe it makes far more sense to modify this line of code than to attempt to abstract all reasonable naming conventions and/or search algorithms.

5 Related Work

Templating engines are certainly nothing new, and are at least as old as the M4 macro processor introduced in 1977 by Kernighan and Ritchie [1]. Using a templating engine to generate web pages has been around since as least the introduction of PHP in 1995 [4]. The ERB templating engine that `KielceRB` uses is part of Ruby on Rails and has been around since 2005 [3].

`KielceRB`'s main contribution is not the templating engine, or the use of a series of data files in a hierarchy; but, rather a minimalist implementation that (1) reduces the steep learning curve and complex installation common in many web frameworks, and (2) makes the tool attractive to those computing instructors who, by their nature, tend to prefer implementing their own tools over conforming their workflow to existing tools.

`KielceRB`'s design is based on an earlier client-side JavaScript tool called simply `Kielce`. When using this client-side version, the browser would load data files from each level of the hierarchy, merge the files into a single data structure, then insert the data into the DOM. The original `Kielce` demonstrated the value of both (1) the hierarchical series of data files, and (2) allowing the data values to be functions; however, the client-side nature of the tool (i.e., inserting the values into documents using JavaScript running in the

browser) proved to be slow, error-prone, and difficult to debug. Overall, it did not provide a good experience for either the instructors or the students [2].

6 Our Experience

We have been using `KielceRB` for two years. It has been a very positive experience: Updating our courses at the beginning of each semester takes less than 10 minutes. Before using `Kielce` it would take up to an hour — not including the time to fix oversights. Preparing new assignments is also very easy. In addition to simplifying the updating of assignments, having a templating engine has made it much easier to give each document a consistent look and feel (with both a common header and a common style sheet).

7 Availability

The software, as well as full documentation and additional sample uses, is available at `https://www.cis.gvsu.edu/~kurmasz/Software`

References

[1] Drian W. Kerninghan and Dennis Ritchie. The m4 macro processor. Technical report, Bell Laboratories, 1977.

[2] Zachary Kurmas. Kielce: Configurable html course documents. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, page 403, New York, NY, USA, 2012. Association for Computing Machinery.

[3] `https://weblog.rubyonrails.org/2005/12/13/rails-1-0-party-like-its-one-oh-oh/`.

[4] `https://groups.google.com/forum/#!msg/comp.infosystems.www.authoring.cgi/PyJ25gZ6z7A/M9FkTUVDfcwJ`.

[5] `https://rubyonrails.org/doctrine/`.