Mantevo Views

A Flexible System for Gathering and Analyzing Data for the Mantevo Project

An Honors Thesis

College of St. Benedict/St. John's University

In Partial Fulfillment

of the Requirements for Distinction

in the Department of Computer Science

by

Cameron Christesen

April, 2007
Abstract

Manttevo is an emerging project in the international high performance computing community. The community of Manttevo users needs a way to easily gather and analyze its data. Our project attempts to fulfill this need by developing a standardized method for gathering Manttevo data as well as by developing a webapp for uploading, viewing, and analyzing that data. The work of this thesis will increase the value and ease-of-use of Manttevo capabilities. Throughout our project, we strove for flexibility in the sense that as the Manttevo project evolves our webapp can adjust with ease. We have been pleasantly surprised to find that our emphasis on flexibility has resulted in a general-purpose solution with possible applications outside the initial scope of the project.

Contents

1 Introduction 4

2 The Data Gathering Process 5

2.1 Standardizing the Format of Manttevo Data . . . . . . . . . . . . 6
2.2 Gathering Multiple YAML Results . . . . . . . . . . . . . . . . . . 6
2.3 Gathering System information . . . . . . . . . . . . . . . . . . . . 8

3 Webapp: Initial Design Considerations 9

3.1 Choosing a Database Model . . . . . . . . . . . . . . . . . . . . . 10
3.1.1 "Flattening" YAML . . . . . . . . . . . . . . . . . . . . . . 10
3.1.2 Finding the Relational Model Superior . . . . . . . . . . . . 11
3.2 Choosing a Programming Language: The Lure of Rails . . . . . . . 11
3.2.1 Attractions . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
3.2.2 Security . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
3.3 Choosing a Database Back-End . . . . . . . . . . . . . . . . . . . 12

4 Result Types 14

4.1 Critiquing Our Approach . . . . . . . . . . . . . . . . . . . . . . . 15
4.2 A Rejected Alternative . . . . . . . . . . . . . . . . . . . . . . . . 15

5 Features 16

5.1 Site Navigation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
5.2 Analysis Tools . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
5.2.1 (HTML) Table Views . . . . . . . . . . . . . . . . . . . . . 17
5.2.2 Detail views . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
5.2.3 Worksheets . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
5.2.4 Google Charts . . . . . . . . . . . . . . . . . . . . . . . . . . 21
5.2.5 Worksheet Charts (Kiviat Diagrams) . . . . . . . . . . . . . 21
5.2.6 Detail Charts (Pie Charts) . . . . . . . . . . . . . . . . . . . 23
5.2.7 XML Generation . . . . . . . . . . . . . . . . . . . . . . . . 23
5.3 User Capabilities . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
1 Introduction

Mantevo is an emerging project in the international high performance computing community. The community of Mantevo users needs a way to easily gather and analyze its data. Our project attempts to fulfill this need by developing a standardized method for gathering data as well as by developing a webapp\textsuperscript{1} for uploading and analyzing that data. The work of this thesis will increase the value and ease-of-use of Mantevo capabilities.

Throughout our project, we strove for flexibility in the sense that as the Mantevo project evolves our webapp can adjust with ease. We have been pleasantly surprised to find that our emphasis on flexibility has resulted in a general-purpose solution with possible applications outside the initial scope of the project.

The Mantevo project provides software packages called miniapplications,\textsuperscript{2} which Mantevo users to “predict performance of real applications in new situations” and “aid computer systems design decisions.”\textsuperscript{3} Running miniapplications (running a package’s executable) generates performance data. In this way, we think of miniapplications as tests that generate results. The term \textit{result} will be used often throughout this document to refer to data gathered from the running of a miniapplication. The term \textit{result type}, which we will also use often, essentially refers to the miniapplication from which a result came.

We embraced a Ruby on Rails solution for our webapp over other alternatives, such as PHP. Ruby on Rails is an open source web development framework written in the Ruby programming language that has gained significant popularity in recent years. Mantevo Views, the name of our Ruby on Rails webapp, was given its name because of the various views of data it provides users. One of the major achievements of Mantevo Views has been to adjust views dynamically for varying result types.

When we embarked on this journey a year ago and Mike Heroux explained the task at hand, I tried to estimate the amount of time it would take to implement a workable solution. A few weeks? A month? Two months? Half a year? All told, the webapp has yet to go live; however, that day draws near, and we have high hopes for our project’s future success and usefulness to the Mantevo community.

\textsuperscript{1}A webapp, or web application, is an application that is accessed through a web browser. Popular webapps include Facebook, Gmail, and Microsoft Outlook Web Access.

\textsuperscript{2}We intentionally misuse the term miniapplication in this document for simplicity’s sake. Mantevo software packages can really be broken down into miniapplications, application proxies, and minidrivers. We use the term miniapplication to cover all three.

2 The Data Gathering Process

“The Mantevo project is an effort to provide open-source software packages for the analysis, prediction and improvement of high performance computing applications.”\textsuperscript{4}

\textsuperscript{4}“About the Mantevo project.”
2.1 Standardizing the Format of Mantevo Data

Prior to the work of this thesis, results were not being collected into an easily-parseable, standardized format. Now, data is collected into the YAML text format.\(^5\) YAML, like XML, which was our original choice, is self-describing. But, unlike XML, YAML is also human-readable (see figure 1). The human-friendliness of YAML and the Ruby programming language’s support of it were our two major considerations when choosing it as our data format.

The next step was to enable existing Mantevo software packages to collect data in the YAML format. We needed to create a simple data gathering tool for miniapplications to use. This goal was achieved by writing two C++ classes (see source code in Appendix D.1 on page 58) capable of building simple hierarchical YAML similar to that shown in figure 1.

It should be noted that we use a subset of YAML’s full capabilities. Our C++ classes only produce YAML of a restricted form. YAML Doc, the first of our classes, contains a tree of objects of the other class, YAML Element. Each YAML Element can have children YAML Elements, but if a YAML Element is a parent, it cannot itself hold any data. This is why, when looking at the sample YAML in figure 1, each parent in the YAML tree holds no data. This is important because it allows us to easily “flatten” results in order to map them into database tables (see section 3.1.1).

The YAML generated by our C++ classes has another important feature besides its simple tree structure. Our webapp, when examining YAML, checks for two specific pieces of data: “Mini-Application Name” and “Mini-Application Version.” YAML Doc requires the name and version of the miniapplication as parameters to its constructor. This is important because result types, as alluded to earlier, are identified by name-version combinations. This will become an important concept later in the discussion.

By standardizing miniapplication output, we have been able to create a webapp capable of handling Mantevo result data. Our webapp expects data of this standardized form; if it receives anything else, it doesn’t know what to do.

Although we have provided the Mantevo project with a tool for gathering data in this way, individual Mantevo packages (miniapplications) are not yet using it. Mike Heroux has created a helpful example of how to use this tool, which can be found in the header file for YAML Doc. Altering existing miniapplications to use our tools can be done by following that example.

2.2 Gathering Multiple YAML Results

In addition to our YAML Doc and YAML Element classes, we created a Perl script (see code in section D.1.5 on page 65) to conglomerate multiple YAML results into a single file. Perl was chosen because of its wide support across platforms. Currently, the webapp does not accept uploads of this primitive conglomerate format, which essentially inserts the text “—SEPARATOR—” in between each YAML result. The script was created because it was thought that

\(^5\)An interesting tidbit: YAML is a recursive acronym for YAML Ain’t Markup Language
Listing 1: YAML

```yaml
---
Mini-Application Name: CS
Mini-Application Version: 2009
school: CSBSIU
faculty:
  qualifications:
    degrees:
      BA: 7
      BS: 1
      MA: 2
      MDiv: 1
      MS: 7
      PhD: 6
    average IQ: 152.566
students:
  average IQ: 152.567
---
```

Listing 2: XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Mini-Application_Name>CS</Mini-Application_Name>
<Mini-Application_Version>2009</Mini-Application_Version>
school: CSBSIU
<faculty>
  <qualifications>
    <degrees>
      <PhD>6</PhD>
      <MDiv>1</MDiv>
      <MS>7</MS>
      <BS>1</BS>
      <MA>2</MA>
      <BA>7</BA>
    </degrees>
    <average_IQ>152.566</average_IQ>
  </qualifications>
</faculty>
<students>
  <average_IQ>152.567</average_IQ>
</students>
```
Mantevo users would be interested in uploading many YAML results at one time; simply put, it would be a nuisance for users to upload each result one at a time through the web interface. In other words, our Perl script was created as a mere convenience.

2.3 Gathering System information

It turns out that system information, such as computer hardware information, is something Mantevo users are very interested in looking at alongside miniapplication result data. We debated the best way to collect and store system information, came up with a solution, implemented that solution, then scrapped it and started over. Our new design is a bit controversial because it leads to data duplication, but it is simpler than our initial design, which didn’t have the data duplication problem.

Our first solution had the benefit of minimizing data duplication. Though we didn’t mention it in the previous section, system information was collected during the running of our Perl script. In the database, a separate table existed for system information, and each result was linked to the system information of its conglomeration file.

We scrapped this solution in favor of a solution that gathers system information during the running of miniapplications themselves. We did this for a few reasons. To start with, the whole point of our previous solution was to try to minimize data duplication, but we weren’t that concerned about data duplication anyway. Our database wasn’t going to grow very large; and furthermore, system information wasn’t going to change after it entered the database, so we weren’t going to run into the classic problem of having to update information in several places. Also, our previous solution went against a fundamental concept of our final system which says that analyzable data should be completely contained within result tables. That is, all information about a given result, including system information, should be stored in the same table. Finally, from a user’s perspective, it’s easier to just run a miniapplication and upload the result file it generates. Running a separate script at a later point in time to gather system information adds complication.

The downside of our new solution is clear. If a user ran 100 miniapplications on the same system and submitted results for each, the same system information would be duplicated 100 times. We’ve decided to tolerate this sort of thing in the interests of simplicity, especially where the creation of new result types is concerned.
3 Webapp: Initial Design Considerations

Before discussing our webapp as it currently exists, some discussion of the evolution of our project and the research behind our design choices is necessary. A year ago, we faced a plethora of design considerations. Before we even chose Ruby on Rails, we addressed questions such as whether or not there was a way for us to preserve the hierarchy in Mantevo result data, what type of back-end storage mechanism we should use, and what programming language we should write our webapp in.
3.1 Choosing a Database Model

In the beginning, all we knew was that we wanted a way to gather, store, and analyze Mantevo results. In the days before we chose YAML, when XML was our data format of choice, we looked into using an XML database, such as Apache Xindice, as the back-end storage mechanism for our Web app. One advantage of an XML database is that, because it stores data as XML, it naturally preserves hierarchy. The relational model, the conventional back-end database model, stores data in flat tables. We saw the loss of hierarchy inherent in the relational model as a cause for concern.

3.1.1 “Flattening” YAML

Despite our concerns, we eventually chose the relational model. But before explaining our reasoning behind that choice, we want to make the idea of loss of hierarchy more clear by showing an example of how we “flatten” YAML results in order to map them into database tables:

<table>
<thead>
<tr>
<th>Listing 3: Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: Obama</td>
</tr>
<tr>
<td>Version: 0.9</td>
</tr>
<tr>
<td>party: democratic</td>
</tr>
<tr>
<td>family:</td>
</tr>
<tr>
<td>parents:</td>
</tr>
<tr>
<td>mom:</td>
</tr>
<tr>
<td>name: Ann</td>
</tr>
<tr>
<td>dad:</td>
</tr>
<tr>
<td>name: Barack</td>
</tr>
<tr>
<td>family:</td>
</tr>
<tr>
<td>mom: Akumu</td>
</tr>
<tr>
<td>dad: Onyango</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 4: Flattened</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: Obama</td>
</tr>
<tr>
<td>Version: 0.9</td>
</tr>
<tr>
<td>party: democratic</td>
</tr>
<tr>
<td>family_parents_mom_name: Ann</td>
</tr>
<tr>
<td>family_parents_dad_name: Barack</td>
</tr>
<tr>
<td>family_parents_dad_family_mom: Akumu</td>
</tr>
<tr>
<td>family_parents_dad_family_dad: Onyango</td>
</tr>
</tbody>
</table>

This flattening process provides an extremely simple way of mapping YAML results into database tables. In the previous example, our app would have

---

6For more information about Apache Xindice, go to http://xml.apache.org/xindice/

7A relational database is composed of tables that interconnect through a variety of relationships. Each table has rows and columns. Each row is a record in the database, and each column specifies a certain piece of information. For example, if we had a table for storing user information, that table might have two columns: name and password. Each row [record] of the table would then be a different user. A relational database does not support hierarchy.
a database table for “Obama version 0.9” with columns named “party,” “family_parents_mom_name,” and so on. But despite the ease of mapping this flattening process provides, the hierarchy of the data is lost.\footnote{8}

### 3.1.2 Finding the Relational Model Superior

The persuasive arguments of professor Imad Rahal in favor of the relational model ultimately convinced us to abandon XML databases. In an email to the author, Dr. Rahal said, “Before the relational model, people used a model known as the hierarchical model. XML is hierarchical by nature so, to me, this seems like a step backwards.”\footnote{Rahal, Imad. "RE: thoughts on an XML database." E-mail to the author. 21 Mar. 2009.} When the relational model was first developed in the 1970s, it “attracted immediate attention due to its simplicity and mathematical foundation.”\footnote{Elmasri, Ramez, and Shamkhan B. Navathe. Fundamentals of Database Systems. 5th ed. Greg Tobin, 2007. Page 145.} Since that time, it has flourished. Today, there are many popular relational database implementations, such as Oracle, SQL Server, and MySQL. It can be reasonably assumed that any webapp one might encounter on the web uses a relational database as its back-end storage mechanism. Our app, Mantevo Views, is no different.

Most web developers don’t even consider using anything other than a relational database for back-end storage. Maybe we missed a unique opportunity to do something new and exciting with XML databases, but probably not. As Dr. Rahal pointed out, the hierarchical model for data storage was ousted by the relational model almost 30 years ago. The hierarchical model, of which XML and YAML are prime examples, is great for data transport, but not necessarily for data storage.

### 3.2 Choosing a Programming Language: The Lure of Rails

“Ruby on Rails is a breakthrough in lowering the barriers of entry to programming. Powerful web applications that formerly might have taken weeks or months to develop can be produced in a matter of days.”\footnote{O’Reilly, Tim. "Quotes." Ruby on Rails. 20 Mar. 2009 <http://rubyonrails.org/quotes> .} – Tim O’Reilly, Founder of O’Reilly Media

At first we leaned towards using PHP because it was well-established and we knew people who had successfully used it. We didn't necessarily see anything wrong with PHP, but catching wind of some hype about Ruby on Rails, we decided to look into it. Tim O'Reilly’s “in a matter of days” comment and others comments like it – comments which testify to Ruby on Rails’s power and simplicity — fueled our excitement about Ruby on Rails. Also, being open

\footnote{We have noted that our flattening process, in a way, still preserves hierarchy through the use of the underscore character ('_'). No attempt has been made to rebuild the hierarchy of results based on underscore characters in column names, though such an endeavor would likely prove fruitful. The reason we have failed to make this attempt is because Mantevo Views, through a different process, already has the capability of rebuilding hierarchy [see section ].}
source, Ruby on Rails was freely available to download and install, so we were able to give it a test run. We liked what we saw.

### 3.2.1 Attractions

A noticeable attraction of Ruby on Rails is the way it interacts with databases. Rails removes the programmer from writing direct SQL,\(^\text{12}\) which not only simplifies the work of the programmer, but if we wanted to move from one relational database implementation to another at some later point in time, say from SQLite to MySQL, it could be done by changing only a couple lines of code. Because Rails allows programmers to write Ruby instead of SQL for querying the database, Rails code is not tied to a specific database implementation.

A year ago, I was just finishing up with CS230, “Software Development.” In the class, I worked as part of a team to developing a software banking system in Java over the course of a semester. The project was large, and despite my group’s best efforts, the final product was shoddy. After running through the sample application found in \textit{Agile Web Development with Rails},\(^\text{13}\) I realized that using Rails I could create the same banking system — a better one actually — in a day, whereas it had taken our group of five programmers an entire semester. This realization warmed me up to Rails in a hurry, and after some discussion with Mike Heroux, we decided to move forward using Ruby on Rails.

### 3.2.2 Security

One of the first things we did with Ruby on Rails was to look into security concerns and evaluate whether or not the Rails framework was secure enough for our purposes. In the end, we found security to be more a test of our abilities as developers than a test of Rails’s abilities as a framework. For Mantevo Views, security concerns come in two main places. First, our app tracks user and admin information — though none of this information is terribly sensitive. Second, our app allows users to upload potentially harmful data. Our main source of information on securing our app has been \textit{Agile Web Development with Rails}. Chapter 27 provides a fairly thorough discussion of security concerns, and we have attempted to follow the advice of that chapter. For more on improving the security of Mantevo Views in the future, see section 9.4.

### 3.3 Choosing a Database Back-End

“It’s a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.”\(^\text{14}\)

\begin{flushright}
 – SQLite.org
\end{flushright}

\(^\text{12}\)SQL stands for “structured query language.” It is used to query relational databases.


As was stated in section 3.2, Rails allows the developer to write database independent code. Rails can work with SQLite, DB2, MySQL, Oracle, Postgres, Firebird, and SQL Server. Having had experience using MySQL, we started with it, but we soon switched to SQLite3. Making this change was exceptionally easy in Rails and required changing only a couple lines of code.

SQLite has advantages and disadvantages when compared with other solutions such as MySQL. SQLite’s advantages — namely its self-contained, zero-configuration, and serverless nature — make it the clear choice for development. However, according to *Agile Web Development with Rails*, “SQLite 3, although excellent for development and testing, isn’t generally regarded as appropriate for large-scale deployment.” MySQL scales much better than SQLite in production due to the fact that an SQLite database “is contained in a single disk file.” This limitation of SQLite makes MySQL the winner in many production environments. However, after consulting Jason Cross, the SJU Linux systems administrator who helped us with an initial deployment of Mantevo Views (see section 8), we believe SQLite will serve our needs fine in production, especially since we don’t anticipate deployment across multiple servers. Should problems arise, switching the production database to a more scalable solution should not be difficult.

MySQL and SQLite are both open source software, which is why they were are two main considerations. Our project uses no proprietary software.

Thomas, page 658.

SQLite Home Page.
4 Result Types

Once we settled on Ruby on Rails, we faced a whole new set of design considerations, such as what features our webapp should offer and what our database should look like. But the biggest question we addressed was how to provide flexibility in the sense that our system could easily handle changes in the data being collected. This last question eventually lead us to the idea of a result type, which we have embraced and built our webapp around.

As was discussed in the data gathering section (section 2), each YAML result holds two fundamental pieces of information: a name and a version. A result type is determined by a name-version combination. Each result type is unique and has its own database table. Just because two result types share the same name (i.e. come from the same miniapplication) doesn’t mean they have to share the same data.

Consider the following example in which we have a fictitious miniapplication named SJU with version 1.0:

Listing 5: SJU 1.0

```
Mini-Application Name: SJU
Mini-Application Version: 1.0
year: 2008
students:
total: 1889
men: 1889
women: 0
```

But imagine a user also wanted to know how many students named Brian attended the school (a reasonable thing to want to know).

Listing 6: SJU 2.0

```
Mini-Application Name: SJU
Mini-Application Version: 2.0
year: 2009
students:
total: 1862
men: 1862
women: 0
Brian: 38
```

This would require adding a new piece of information to each YAML result. But there’s a problem: version 1.0 is restricted to collecting a limited set of information; namely, the year, the total number of students, the number of men, and the number of women. In our database, the table for SJU 1.0 is rigid by choice; we don’t allow new columns to be added to result tables. Because of this, our method of accommodating change is to create whole new result types by changing the versions of miniapplications. Hence, in our example, when we decide we want to start collecting information about the number of Brians that
attend SJU, we need to assign the SJU miniapplication a new version (version 2.0).

It should be noted about this example that, although data for SJU 2.0 is similar to that of SJU 1.0 this needn’t be the case. There is nothing that says SJU 2.0 and SJU 1.0 need to have any data whatsoever in common. SJU 2.0 could just as easily have looked like the following:

Listing 7: SJU 2.0 Could Look Like This

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mini-Application Name: SJU</td>
</tr>
<tr>
<td>Mini-Application Version: 2.0</td>
</tr>
<tr>
<td>monks: 89</td>
</tr>
<tr>
<td>prayers per day: 22081.344</td>
</tr>
</tbody>
</table>

Changes to miniapplications are inevitable. New miniapplications will be created and old ones will change versions. In response, we need to be able to create new result types easily. This process is described in section 7.

4.1 Critiquing Our Approach

Our use of result types has a couple of downsides. To start with, it has the potential to create lots and lots of database tables. Is this a bad thing? We’re not so sure, but it seems to stray away from our ideal of simplicity. Secondly, because result types are so fixed – any small change to the data collected by a miniapplication requires a change of version and thus the creation of a new result type – we run into the problem of not being able to compare data across result types. For example, though we might want to view data from SJU 1.0 and SJU 2.0 side-by-side, Manttevo Views currently provides no facility for doing so. In the future work section (section 9.5), we provide an idea for how this limitation might be overcome.

On the positive side of things, we don’t expect miniapplications to change much. If we’re right, the previously mentioned concerns may be effectively nullified.

4.2 A Rejected Alternative

From a database perspective, we could have chosen to create one table for each miniapplication, as opposed to our current approach which creates one table for each version of each miniapplication. To accommodate changes in the data gathered for each miniapplication we would have had to develop a system of adding columns to miniapplication tables. The fundamental problem with this approach is that different versions of the same miniapplication could be collecting radically different data. This would lead to a database table with lots of NULL values. At the time we rejected this solution, we didn’t have a way of figuring out what columns belonged to what versions, which made figuring out what information to display back to users difficult. Now, we have such a method, and we’re thinking about going back to this rejected alternative. The
core idea of a result type would stay intact, but the way data is stored in the
database would change.

The main reason we would want to go back to this rejected alternative is
that it provides us with a way to more easily compare result data across different
versions of the same miniapplication. This is only one of our ideas for achieving
this functionality. Our future work section (section 9.5) discusses these ideas in
more depth.

5 Features

With discussions of the data gathering process and result types behind us, we
can move on to discussing Mantevo Views itself. We’ll begin with an overview
of the features of Mantevo Views.
5.1 Site Navigation

In order to view the data they upload, users need a way to easily navigate between miniapplications and versions. Mantevo Views provides a navigation menu at the top of each page. Each menu item is a different miniapplication. Hovering over a menu item displays a drop-down sub-menu of links to versions of miniapplications.

5.2 Analysis Tools

Mantevo Views functions as a storage facility allowing users to upload and view their data as well as everybody else's. Besides just storing data, Mantevo Views provides basic analysis tools. It derives its name from the various views of data it presents to users. Of particular interest are Mantevo Views’s use of worksheets and charts as analysis tools.

All examples shown in this section use fictitious result data of the form shown in the example of the YAML Doc header file (see code in section D.1.2). A YAML result of this type would look like the following:

```
Listing 8: SJU 2.0 Could Look Like This

---
Mini-Application Name: HPCCG
Mini-Application Version: 1.0
final_residual: 1.4523e-13
time:
  total:
    time: 2.457
  flops: 4.88e5
ddot: 1.243
sparsemv:
  time: 0.3445
overhead:
  time: 0.0123
percentage: 0.034
---
```

5.2.1 (HTML) Table Views

Mantevo Views provides, as its most basic analysis tool, HTML tables for viewing and sorting data. Each table view holds only results of one result type. These tables are sortable on every column (see figure 3). Paging is also provided. The current default for paging is 20 results per page.

Table views will change depending on the following criteria: whether or not the user is logged-in; and, if the user is logged-in, whether or not the user is viewing data in a worksheet (worksheets are described in section 5.2.3). If the user is not logged-in, a plain HTML table will be displayed (see figure 2). If the user is logged-in, checkboxes will be displayed alongside each row of the table allowing a user to add results to a worksheet (see figure 4). If a user is logged-in and using a worksheet, similar checkboxes will be displayed, but this time to allow for removal of results from the worksheet (see figure 5).
<table>
<thead>
<tr>
<th>ID</th>
<th>SparseMV Overhead Time</th>
<th>DDOT Time</th>
<th>SparseMV Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0123</td>
<td>1.243</td>
<td>0.3445</td>
</tr>
<tr>
<td>2</td>
<td>0.0314</td>
<td>1.4557</td>
<td>0.2987</td>
</tr>
<tr>
<td>3</td>
<td>0.067878</td>
<td>1.01445</td>
<td>0.3545</td>
</tr>
<tr>
<td>4</td>
<td>0.075465</td>
<td>1.45511</td>
<td>0.4355</td>
</tr>
<tr>
<td>5</td>
<td>0.05456</td>
<td>1.2124</td>
<td>0.0145</td>
</tr>
<tr>
<td>6</td>
<td>0.0254</td>
<td>1.8989</td>
<td>0.5678</td>
</tr>
<tr>
<td>7</td>
<td>0.3022</td>
<td>1.0445</td>
<td>0.3333</td>
</tr>
</tbody>
</table>

Figure 2: HTML Table (standard view)

<table>
<thead>
<tr>
<th>ID</th>
<th>SparseMV Overhead Time</th>
<th>DDOT Time</th>
<th>SparseMV Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.05456</td>
<td>1.2124</td>
<td>0.0145</td>
</tr>
<tr>
<td>2</td>
<td>0.0314</td>
<td>1.4557</td>
<td>0.2987</td>
</tr>
<tr>
<td>3</td>
<td>0.067878</td>
<td>1.01445</td>
<td>0.3545</td>
</tr>
<tr>
<td>4</td>
<td>0.075465</td>
<td>1.45511</td>
<td>0.4355</td>
</tr>
<tr>
<td>5</td>
<td>0.0254</td>
<td>1.8989</td>
<td>0.5678</td>
</tr>
</tbody>
</table>

Figure 3: HTML Table (sorted on a column)

<table>
<thead>
<tr>
<th>ID</th>
<th>SparseMV Overhead Time</th>
<th>DDOT Time</th>
<th>SparseMV Time</th>
<th>Checkbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0123</td>
<td>1.243</td>
<td>0.3445</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>0.0314</td>
<td>1.4557</td>
<td>0.2987</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.067878</td>
<td>1.01445</td>
<td>0.3545</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.075465</td>
<td>1.45511</td>
<td>0.4355</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>0.05456</td>
<td>1.2124</td>
<td>0.0145</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.0254</td>
<td>1.8989</td>
<td>0.5678</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.3022</td>
<td>1.0445</td>
<td>0.3333</td>
<td></td>
</tr>
</tbody>
</table>

Choose worksheet
My Worksheet
Add To Worksheet

Figure 4: HTML Table (logged in user view)
5.2.2 Detail views

In each HTML table view, a link to a detail view is provided. Detail views exist to show the user all of the data associated with a given result. HTML table views in general only show a subset of the complete data. We did this to avoid having HTML tables with too many columns. Detail views come in three forms: flat tables (see figure 7), original YAML (see figure 6), and charts (see figure 8). Flat tables show all pieces of data as they are stored in the database, while original YAML simply displays the YAML (hierarchy and all) that was originally uploaded to Mantevo Views. Detail charts are discussed in section 5.2.6.
Figure 7: Detail View (flat table)

Figure 8: Detail View (charts)
5.2.3 Worksheets

Worksheets serve as collections of results. Each worksheet can store results of every result type; however, worksheets group results according to their type. In other words, each table view in a worksheet displays only results of the same result type.

There are two main benefits of worksheets. Most importantly, worksheets allow users to organize the results they are interested in. These results may have been uploaded by themselves or by other users. The second main benefit of worksheets is their ability to produce charts.

5.2.4 Google Charts

After testing out various charting tools, especially those written in Ruby, we settled on the Google Chart API. The idea behind the Google Chart API is simple: a user embeds information about his or her desired chart into a URL, and Google's servers produce a chart in the PNG (Portable Network Graphics) image format. All a user of the API needs to do is place such a URL into an HTML image tag. Mantevo Views embeds information into URLs with the help of the google_charts_on_rails plugin for Rails.

5.2.5 Worksheet Charts (Kiviat Diagrams)

Worksheet charts are an analysis tool we think Mantevo users will find especially useful. Users create their own worksheet charts, deciding exactly what data they are interested in analyzing. Mantevo Views saves this information and uses it to create radar charts. The idea behind such a chart, as illustrated in figure 9, is to compare results (represented by colored lines) across attributes (the spokes of the chart). These charts, once created, automatically appear in a worksheet. If a user adds or removes results from a worksheet, worksheet charts update automatically. To aid in interpreting charts, a table of the data used in generating the chart is shown beside the chart itself.

Each worksheet chart belongs to a user but not a worksheet. The implication is that when a user creates a worksheet chart, that chart will be available to him or her not only in that worksheet, but in any worksheet. This was a design choice which could easily be changed. We realize that it might make more

---

18Google doesn’t really place a limit on the number of charts that can be generated each day: “There’s no limit to the number of calls per day you can make to the Google Chart API. However, we reserve the right to block any use that we regard as abusive. If you think your service will make more than 250,000 API calls per day, please let us know by mailing an estimate to chart-api-notifications@google.com.” This quote was taken from “Developer’s Guide - Google Chart API -,” Google Code. 29 Mar. 2009 <http://code.google.com/apis/chart/>.

19Plugins such as the google_charts_on_rails plugin are found in the ‘vendor’ directory of the rails project.

20It was necessary to edit the google_charts_on_rails plugin code for use with worksheet charts in order to support multiple data sets. See <http://code.google.com/p/google-charts-on-rails/issues/detail?id=8> for the patch we used.

21Another name for radar charts is kiviat diagrams, but since Google charts uses the term radar chart, we will too.
### Worksheet: My Worksheet

<table>
<thead>
<tr>
<th>ID</th>
<th>SparseMV Overhead Time</th>
<th>DDOT Time</th>
<th>SparseMV Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>0.075645</td>
<td>1.45511</td>
<td>0.4355</td>
</tr>
<tr>
<td>01</td>
<td>0.0123</td>
<td>1.248</td>
<td>0.3445</td>
</tr>
<tr>
<td>03</td>
<td>0.067878</td>
<td>1.01445</td>
<td>0.3545</td>
</tr>
</tbody>
</table>

**Create/edit charts**

<table>
<thead>
<tr>
<th>ID</th>
<th>DDOT Time</th>
<th>Total Time</th>
<th>SparseMV Time</th>
<th>Final Residual</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1.45511</td>
<td>2.2454</td>
<td>0.4355</td>
<td>1.2345e+12</td>
</tr>
<tr>
<td>1</td>
<td>1.243</td>
<td>2.457</td>
<td>0.3445</td>
<td>1.4523e+13</td>
</tr>
<tr>
<td>3</td>
<td>1.01445</td>
<td>2.66755</td>
<td>0.3545</td>
<td>9.674323e+12</td>
</tr>
</tbody>
</table>

**Figure 9: A Worksheet with a Chart**
intuitive sense for a worksheet chart to belong to a worksheet. The advantage of having a worksheet chart belong to a user is that this may save the user from having to make the same charts in every worksheet.

5.2.6 Detail Charts (Pie Charts)

Detail charts, like worksheet charts, are created by individual users, but instead of radar charts, these charts are simple pie charts. These charts are not tied to a specific result but to a specific result type. Whenever a user examines the detail view of a result, Mantevo Views generates all detail charts that user has created for that result’s result type. In the case of figure 8, the user had created two charts for the result type of the result shown. Notice that the first chart in figure 8 doesn’t make much sense; it’s an inappropriate use of a pie chart. But, the following chart is actually quite useful and shows a simple breakdown of performance data.

5.2.7 XML Generation

Even though YAML pervades Mantevo Views, we’re aware that some users might want to receive data in the XML format. Because of this, a link is provided on every page containing table view to request data in XML form. Users will likely take this data and use it to carry out further data analysis beyond what Mantevo Views can supply. It should be noted that this XML contains results with the hierarchy of the original YAML results preserved. This is accomplished by converting original YAML results, which we store in our database as text, into XML. Storing original YAML means that we have some data duplication in each result table, but storing the original YAML has proven to be very useful. Not only does it preserve hierarchy for generating XML, but it also provides a very nice detail view of data, as shown in figure 6.

5.3 User Capabilities

Anyone who wishes to become a user of Mantevo Views can go through a simple registration process. All that this process requires is supplying an email and a password. Once registered, users can log in, upload results, create worksheets, and create charts.

The login process is simple: a user provides an email and password which Mantevo Views validates. We implemented a smart login system using a session. Consider the case where a user is not logged-in and tries to access a restricted page, such as a worksheet page. Mantevo Views redirects that user to the login screen and stores the URL of the requested page in the session. Once logged-in, the app redirects the user back to the originally requested page.

Users can upload new results. On the upload page, a user is asked to select a file. If the file uploaded is proper result YAML for an existing result type, the result is saved into the database; otherwise, the user is send an error message.
Users are allowed to delete results they themselves have uploaded, but nobody else’s.

Users can create worksheets (worksheets are described in section 5.2.3). All that is required is for the user to name the worksheet. Users can delete worksheets they have created.

Users can create worksheet charts (see section 5.2.5) and detail charts (see section 5.2.6). The process is similar for both. First, a user must provide a name for the chart. Then, a user must select the data he or she would like to have displayed in a chart. Users can delete charts they have created.

### 5.4 Admin Capabilities

Admins are provided with the same type of login and authentication system as users. Once logged in, admins have the power to create new result types and edit existing result types. The future work section discusses other powers admins could have (see section 9.11).

### 5.5 Adding a New Result Type: The Admin Upload Process

This section only discusses the process from within Mantevo Views itself. A description of the entire process, including redeploying the webapp once the changes have been made, can be found in appendix B.

Once logged-in, admins can upload a YAML result of the result type they wish to add to Mantevo Views (see figure 10). The admin is then given the ability to alter the names generated from the uploaded result YAML (see figure
The reasoning for this is that generated names are difficult to read and generally contain a lot of underscore characters. Furthermore, generated names are usually so long that, because these names are used in table views and charts, they destroy the functionality of Mantevo Views. Finally, the admin is given the choice of what columns to display in an HTML table view (see figure 12).

When this process is complete, a single YAML file is generated into the results directory of the app. Interestingly, the admin upload process takes YAML in and gives YAML out. At this point, Mantevo Views still knows nothing about the new result type.

5.6 Finish the Job with Rake

We have created a Rake task to take the output of the admin upload process and update the webapp to handle the new result type. It generates files, inserts code, and creates new database tables all in a single step. We have also created a Rake task to remove result types. Our code can be found in the application at /lib/tasks/result.rake. The code is also in the Appendix section E.5. A more technical discussion of creating new result types is given in section 7.

5.7 Altering a Result Type

From an admin’s perspective, altering a result type is just like creating a new result type. Once logged-in, an admin can select the desired result type to edit. The admin is then walked through a process similar to that described in section 5.5. The difference is that no file is generated and no Rake task needs to be run. Changes take effect immediately.

---

Rake is a tool similar to its “grandpa” Make in that it’s used to create tasks and has dependency tracking, but Rake is less cryptic and a whole lot easier to use. An article entitled Ruby on Rails Rake Tutorial (aka. How rake turned me into an alcoholic) on RailsEnvy.com provides a great introduction to Rake. Rails makes use of Rake to do everything from creating database tables to generating documentation. Rails also allows developers to create their own Rake tasks, which is what we did.
6 Behind the Scenes

Understanding the database is essential to understanding Mantevo Views. This section aims to describe database tables and their relationships. See figures 13, 14, and 15 for an overview of the database structure.
Figure 14: ResultType

Figure 15: Admin
6.1 Result Tables

Though figure 13 shows entities revolving around the USER entity, RESULT entities (i.e. result tables\(^{23}\)) are at the true heart of Mantevo Views. They store the data Mantevo users gather, upload, and analyze to improve their systems. The question marks in figure 13 represent the fact that miniapplications can collect whatever data they want. Each result table also has a column for storing the original YAML that users uploaded.

Although figure 13 shows only one RESULT entity, technically there is one for each result type. Mantevo Views maps uploaded results into appropriate result tables based on what “Mini-Application Name” and “Mini-Application Version” it finds in each result. Though there are many RESULT entities, they each relate to the USER and WORKSHEET entities in the way depicted in figure 13.

Each result table in the database is named after the version of the miniapplication for which it stores data.\(^{24}\) For example, the fictitious miniapplication Teacher version 1.0 would be named something like TeacherV1point0.\(^{25}\) The names of tables are important because of the way in which Mantevo Views dynamically keeps track of result tables via the ResultType table described in the following section.

6.1.1 The ResultType Table

Figure 14 shows a standalone entity named RESULT TYPE. This entity has no relationship to any other entity in the database. Its sole purpose is to store information about the result tables of the database. The ResultType table has one entry (row) for each version of each miniapplication; that is, for each result type. These entries are created by our Rake task.

We can separate the attributes of the RESULT TYPE entity (the columns of the ResultType table) into two groups. The first group – name, app, and version – is responsible for holding the name of a given result type. By examining the contents of the ResultType table, Mantevo Views knows all of its result tables. Of the three attributes – name, app, and version – name is the most important because it is the real name of a result type. The app and version attributes are only necessary for setting up pretty site navigation.

\(^{23}\)We’re using the terms table and entity interchangeably here. We recognize that the term entity is usually reserved for discussing ER diagrams and that entities don’t necessarily correspond to tables in the physical implementation of the database. It happens to be the case for our database that entities correlate directly with tables, hence we use the terms interchangeably.

\(^{24}\)The code for generating names based on a given miniapplication and version can be found in appendix section E.8.1 of this document. Within Mantevo Views, the code can be found in /lib/result_helper.rb, method generate_name.

\(^{25}\)In Rails there is a difference between table names and model names. Rails models wrap database tables. TeacherV1point0 is technically a model name. A Rails developer uses model names when writing code to refer to tables. In the same way, we will be using model names in this document to refer to tables instead of using actual table names.
The second group is responsible for holding three data structures: a hash named Flat YAML, a hash named Alternate Names, and an array named Columns. Each of these is stored in the database as YAML text. These structures allow Mantevo Views to generate its views dynamically depending on result type. For example, if Mantevo Views wants to display an HTML table of results from a result table, it has to know what columns to display. Maybe a given result table tracks 50 pieces of data; should an HTML table really display all 50 columns? Even if we wanted it to, we need a way to dynamically access column name information; that is, we don’t want to write separate code for each result type. In the same way, when a user uploads result YAML to Mantevo Views, we need to have a way of dynamically mapping data into the appropriate columns of the appropriate result table. These problems are solved by storing Flat YAML, Alternate Names, and Columns in the ResultType table.

Flat YAML is a hash whose keys are the columns of a result table that point to the data that was present in the YAML result used during the result type creation process (see section 3.1). It’s the least important of our three data structures because its most interesting information – its keys being column names of a result table – is also stored in Alternate Names. The only real use of Flat YAML is in creating charts. Mantevo views can store integers, floats, and strings in result tables. When creating a chart, we don’t want to give users the option of using string data. We use Flat YAML to remove string data as

26 Currently, the process of creating a new result type automatically converts all integers to floats, so each result table actually only stores floats and strings.
an option.

Alternate Names is a hash whose keys are the column names of a result table, just like Flat YAML; but, these keys point to “alternate names” (see an example in figure 16). Alternate names are used to clean up the way data is displayed back users. While it is acceptable use a name like “time_sparsemv_overhead_percentage” for a database column name that nobody sees, it is not acceptable to use this long, ugly name as a table header in a table view or as a label in a chart. “Alternate names” are shorter, prettier names that are used for display purposes.

The column name information stored in Alternate Names and Flat YAML – our code uses Alternate Names, but either would work – is used to dynamically map incoming results into result tables. Uploaded YAML results are flattened into a hash. The keys of this hash should match the keys of Alternate Names and Flat YAML. Using these keys, we map each piece of data into an appropriate column.

Columns, our last data structure, simply stores an array of column names (see figure 17). These names specify the result table columns that are used in table views.

6.2 Charts

Looking at the ER diagram, we can see that the attributes of WORKSHEET CHART and DETAIL CHART are exactly the same, so why have different tables? Couldn’t we have just used one and had another attribute to specify the chart type (detail or worksheet)? The two entities could have been combined into one, but there doesn’t appear to be any real advantage to this. Furthermore, separating the tables seems like a wise idea if more charts are going to be made in the future (see section 9.6).

The two interesting attributes of each chart entity are URL and Columns. Columns is basically the same as Columns for the ResultType table; it’s an array of the column names we want in the chart. URL is a “fake” Google chart URL in the sense that it is used only for display purposes when creating charts. Real Google chart URLs are generated dynamically and aren’t stored in the database.

6.3 Admins and Uploads

The ADMIN entity and the UPLOAD entities, as depicted in figure 15, share a one to one relationship; each admin has one upload.\footnote{We probably could have come up with a more appropriate name than upload for the Upload table, but the reason we named it as we did was that admins upload a result at the beginning of the process for creating a new result type.} The Upload table store exactly the same sort of data as the ResultType table. In fact, through the process of creating a new result type, the data of the Upload table will essentially transfer over to the ResultType table.

The Upload table was created to solve the problem of storing too much information in a session. We use sessions in Manzello Views to store, for example,
user IDs.\textsuperscript{28} This works because a user ID takes very little space; however, we can’t store new result type information in sessions because sessions are not large enough. Also, conventional wisdom says that storing lots of data in sessions is a bad idea anyway.

\textsuperscript{28}A user ID is the primary key of the User table. Rails defaults to using integers for primary keys of all database tables. We use a user ID to query the User table for more information on a user.
Figure 18: MVC Architecture (image taken from Thomas, p 23)

7 Creating New Result Types

In Mantevo Views each result type (each version of each miniapplication) is its own entity. For example, HPCCG version 1.0 would be completely distinct from HPCCG version 2.0. Each result type brings with it extra code to Mantevo Views, though thanks to our recent achievements in code refactoring, each result type requires much less code than it did before. At first, we envisioned writing code for each result type manually. We have since developed an automated system that generates code for us. Generating code has the advantages of being easier, faster, and less error prone.
7.1 Result Types and Rails’s MVC Architecture

The code specific to a given result type can be thought of in terms of Rails’s MVC (model, view, controller) architecture (see figure 18). Rails forces developers to adhere to the MVC architecture. In Ruby on Rails, models wrap database tables, views generate HTML, and controllers “orchestrate the application.” A controller receives a request from a user, uses models to draw information from the database, processes that information, and finally displays an appropriate view to the user.

Mantevo Views has a model for each result type because it has a table for each result type. Likewise, each result type has its own set of views and a controller that handles everything from creating charts to displaying table views for that result type. This is a significant amount of code associated with each result type.

7.2 Staying DRY

“DRY stands for don’t repeat yourself – every piece of knowledge in a system should be expressed in just one place. Rails uses the power of Ruby to bring that to life. You’ll find very little duplication in a Rails application; you say what you need to say in one place – a place often suggested by the conventions of the MVC architecture – and then move on.”

The DRY principle is very closely related to the software development principle of modularity. Xiaoping Jia, author of *Object-Oriented Software Development Using Java*, expresses the idea of modularity by asserting that “A complex software system should be decomposed into a set of highly cohesive but loosely coupled modules.” Jia goes on to say that “Decomposition of complex software systems into modules is one of the most intriguing tasks in software development and is more an art than a science.” We couldn’t agree more. Our attempts to employ modularity and stay DRY lead to our most interesting challenges and our greatest achievements, especially where result types are concerned. Some tricks of the Ruby programming language were fundamental in allowing us to stay DRY.

7.3 Code Generation: Initial Approach

We observed that creating a new result type was merely a matter of generating MVC code. We also observed that if we knew the structure of a YAML result of a new result type, we had all of the information we needed to generate this code. So, we devised a system that examined a YAML result and generated MVC code.
based on the structure of that result. This solution was straightforward, but there was a problem with it: it generated lots and lots of code.

At its height, it generated approximately 1000 lines of code across 15 files for each new result type. Not only was this making Mantevo Views extremely bulky, but all of the generated code was so similar that it begged for code refactoring. Furthermore, we were concerned over what would happen if we wanted to make changes to the way Mantevo Views handles results in the future. For example, suppose we came up with a new analysis tool and wanted all result types to have this feature. We were alarmed to realize that implementing this new feature would mean changing the code for each and every result type.

We have solved this problem in an interesting way. Though our method may be somewhat complicated, the overall result is code simplicity. In our new method, we generate 3 regular files (2 models and a controller for a total of 15 lines of code), 2 migration files (about 70 lines of code total), and we insert code into 3 pre-existing files (a total of 15 lines of code).

7.4 Code Generation: Current Approach

Our final solution revolves around two key ideas. Firstly, we can use inheritance. All result controllers are subclasses of a parent result controller, and all real work is done in the parent class. The result controller subclasses that we generate for each result type are hollow shells; they are two lines long and add absolutely nothing new to what they have inherited from the parent class. Secondly, we can use our ResultType table that holds information about each result type to allow the parent class to act in distinct ways depending on what subclass is being used.

To utilize the ResultType table, the parent result controller uses Rails routing to know what version of what miniapplication its dealing with. Rails handles routing of HTTP requests in a simple way. Consider a the following request URL: ...

But, cleaning up subclass result controllers is only half of the story; our final solution also generates far fewer files. This is because, now that we can dynamically handle differences between result types, there is no need to write new Rails views for each new result type.

In the end, our current approach is very DRY, which has profound implications for future maintenance of the app. Changes and improvements to Mantevo Views can now be made in a single place in the code.
7.5 Ruby Class Names are Constants: A Nifty Trick

There is one interesting component of this system that deserves mentioning. All that the parent result controller initially knows at the beginning of a request is a result type name (e.g. “hpccg_v1point0”), but it needs access to the model for that result type in order to query the database. Normally, a Rails developer could just write code like the following to access results from HPCCG 1.0: “HpccgV1point0.find_all.” But, all the parent result controller has at this point is a string which says “hpccg_v1point0,” so that’s not an option. It turns out that it’s possible to access the model class dynamically using that string. The solution relies on the fact that in Ruby “class names are constants.” [...]

The parent result controller converts the string it has into a constant of the correct form in one line of code: “result_type.camelsCase.constantize” (see the parent result controller in /app/controller/result_controller.rb). This constant, which serves as a class name for a Rails model, can then be used to query the database as usual.

The importance of this trick is that it completely removed all result type specific code from our controllers and views. It was the last piece of the puzzle for staying DRY.

7.6 Altering Existing Result Types

Since all of the information about a given result type is essentially stored in the ResultType table, making changes to the way a result type performs can be done without changing any code. All that needs to be changed is the contents of the ResultType table. The only information stored by the ResultType table which can be altered by an admin are the Columns array and the Alternate Names hash. Altering this information changes the HTML table views. Changes to Alternate Names also appear in charts and detail tables.

---

8 Deployment with Capistrano

"Deployment is supposed to mark a happy point in the lifetime of your application. It’s when you take the code that you’ve so carefully crafted and upload it to a server so that other people can use it. It’s when the beer, champagne, and hors d’oeuvres are supposed to flow."

With the help of Jason Cross, the SJU Linux networks administrator, have successfully deployed earlier versions of our webapp, but these deployments were intended only to test the waters of the deployment process. Jason Cross configured our deployment system to use an Apache Web Server, a Mongrel Rails server, a Subversion repository, and an SQLite database. Our deployment process is facilitated by the use of Capistrano, a tool which promises “turn-key deployment of Rails applications.”

9 Future Work

The intense code refactoring of the past three months has not created a perfect system devoid of any need for improvement. Rather, it has created an improved system whose good coding practices welcome future improvements.

34 Thomas, page 658.
9.1 Mantevo GUI

Future work in improving Mantevo’s data gathering system might aim at presenting Mantevo users with an easy-to-use GUI (graphical user interface). Kurtis Nusbaum, a current SJU student, was working on creating a GUI for the Traman to project, and we had hoped that we could use some of the knowledge he’d gained about creating GUIs with Qt, an open source GUI creation tool, to make a GUI for Mantevo. Such a GUI, if it were to exist, would simplify the process of using Mantevo software packages. From within the GUI, users could compile and run miniapplications. The GUI might also allow results to be conglomerated at the click of a button. It also seems possible that, after running a miniapplication, uploading data to the webapp could be done at the click of a button from within the GUI.

9.2 The Perl Script (Gathering Multiple Results)

If a multiple file upload (see section 2.2) is deemed necessary in the future, a few things should be done. Firstly, the result gathering script should not collect system information. Secondly, the script should collect results – results are YAML text – into a YAML array, not YAML separated by “-SEPARATOR-,” which is the caveman solution the script currently uses. YAML can handle an array structure, and this new conglomeration format would essentially contain an array of YAML results while still being true YAML itself. Lastly, the the webapp would need to be edited to accept such YAML. This would involve tweaking the existing code to iterate over an array of results instead of handling a single result.

9.3 Testing

Rails provides great support for writing tests – unit, functional, and integration. So far we have underutilized these capabilities. This needs to change. One of the reasons testing has been underutilized is the dynamic nature of the way Mantevo Views handles results. We don’t want to pollute the application by generating tests for each new result type that enters the system. Instead, we should find a way to make use of the contents of the ResultType table to dynamically test every result type in the database. These tests should be run automatically at the end of our Rake task (see code in the Appendix section E.5).

9.4 User Authentication and Security

The current authentication for users and admins is rather simple and was taken, with slight modifications, from Agile Web Development with Rails. At this time, there really isn’t any sensitive data being stored in the database, so security concerns are somewhat less than they otherwise would be. Nonetheless, an obvious area of improvement and further research is authentication. We would suggest looking into the restful _authentication plugin for Rails as a place to
start. It would be nice to have a registration system that involved the sending of a confirmation email to finalize the initialization of a user account.

We have developed Mantevo Views with some security issues in mind, but there is certainly room for improvement. We recommend looking to the Ruby on Rails Security Project.\textsuperscript{36}

9.5 A Worksheet View Across Result Types

Worksheets currently serve to allow users to conglomerate their favorite results for a specific result type. While this works fine to analyze results of the same type, users may want to compare results across types. For example, it seems reasonable that a user might want to compare results of SJU 0.8 and SJU 0.9 (two fictitious result types) in the same table. We propose several ideas for solving this problem. The first idea involves using like column names. If it turned out that different versions of the same miniapplication shared almost all of the same column names, this idea might work well. The second idea seems to have a lot more potential than the first. This idea is a little bit more complex, but it would allow for the creation of what we would call “result groups.” The third idea is just an extension of the first two ideas. All it says is that we should consider using Alternate Names (stored in the ResultType table) instead of the actual column names. Finally, our fourth idea involves considerable changes to the basic structure of the database. It involves going back to an old idea of having one table for each miniapplication instead of one table for each version of each miniapplication (see section 4.2). This idea, if implemented, would probably need to be done before the webapp goes live. The other ideas could easily be implemented while Mantevo Views is in production because these ideas don’t alter the structure of existing result tables.

9.5.1 Idea 1: Matching Column Names

Provided that at least some columns of disparate result tables shared the same names, it seems that it would be possible to create an HTML table view showing results of multiple types. This column overlap, if it were to exist, makes us think there might be a good way to use worksheets, which can store results of all types but currently group results according to type, to allow users to analyze data across types. Code would need to be written to compare the columns of multiple result tables (this information is in the ResultType table). Somehow, based on matching column names, a table could be generated showing data from multiple result types.

9.5.2 Idea 2: Result Groups

This idea is significantly more complicated than the first, but it seems much more promising. It would likely involve the creation of another database table. This new table, ResultGroup, would share a many to many relationship with

\textsuperscript{36}Ruby on Rails Security Project. \url{http://www.rorsecurity.info/}.
the ResultType table. A group would have a name and all of the necessary column information about different result types to be able to compare results of disparate result types:

A big difference between this idea and the first idea of matching column names is that now someone – maybe a user, maybe an admin – can decide how to link result types together. This is a major advantage. Imagine the scenario where two result types SJU 0.8 and SJU 0.9 track nearly all of the same student information, but the hierarchy structure of the result YAML had been changed between versions:

**Listing 11: SJU 0.8**

<table>
<thead>
<tr>
<th>MINI APPLICATION NAME: SJU</th>
<th>MINI APPLICATION VERSION: 0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>students: 1876</td>
<td></td>
</tr>
<tr>
<td>breakdown:</td>
<td></td>
</tr>
<tr>
<td>men: 1876</td>
<td></td>
</tr>
<tr>
<td>women: 0</td>
<td></td>
</tr>
</tbody>
</table>

**Listing 12: SJU 0.9**

<table>
<thead>
<tr>
<th>MINI APPLICATION NAME: SJU</th>
<th>MINI APPLICATION VERSION: 0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>students: total: 1889</td>
<td></td>
</tr>
<tr>
<td>men: 1889</td>
<td></td>
</tr>
<tr>
<td>women: 0</td>
<td></td>
</tr>
</tbody>
</table>

In this example, even though the two result types track exactly the same type of data, the YAML structure has changed, and with our current YAML flattening process, the column names in the database will be different for each result type. Result groups would overcome this problem by allowing comparable columns to be linked together in some way.

### 9.5.3 Idea 3: Use Alternate Names

It might be possible to use Alternate Name from the ResultType table in the same way that it was suggested using database column names in the previous
two ideas. It should be noted that while result table column names are fixed once a new result type has been created, alternate names can be changed by admins.

9.5.4 Idea 4: One Table Per Miniapplication

This idea was introduced in section 4.2. Instead of having one result table for each version of each miniapplication, there would be one table per miniapplication to hold data for all of that miniapplication's versions. The idea of a result type, however, would still apply, but the database implementation would change. In this new database implementation, new versions of miniapplications might add new columns that older versions didn't have. We would use our ResultType table to know which columns were used by which result types.

At this point, this idea starts to look a little like idea 1 in the functionality it provides. If we implemented this idea, we would be able to look at versions of a miniapplication, see what data they share in common, and generate HTML table views accordingly. Using this database design, we could also implement idea 2, or 3, or both.

Implementing this solution would require restructuring a fair amount of the app. Before doing so, we would want to be convinced that this solution would add considerable functionality. So far, we're unconvinced. We like the fact that this approach limits the number of database tables, but we don't like the fact that miniapplication tables would likely have a lot of NULL values, especially as time progressed and many new versions of miniapplications were developed.

To make this idea work to its fullest potential, Mantevo users, as much as possible, should refrain from changing the structure of a YAML result from version to version. The sorts of changes illustrated in section 9.5.2 would lead to creating different column names for essentially the same piece of data. This would defeat the purpose of having one table per miniapplication and would litter the miniapplication table with NULL values.

9.6 More Charts

As Mantevo Views stands, there are only two types of charts being generated: worksheet (radar/kiviat) charts and detail (pie) charts. Using the same charting system, new types of charts could be for created for the detail view of data, the worksheet view of data, or both. If a view across result types, as described in section 9.5, were created, charts could be generated for such a view as well.

9.7 Better Worksheet Charts

An HTML table is displayed beside each worksheet chart (different from the regular table view). Such a table shows the data that was used in the making of the chart. This is important because our worksheet charts don't display numbers on them. However, it's a little difficult to compare back and forth between radar charts and tables. The comparison could be made easier if the background colors
of each row of the table matched the color of the corresponding result (colored line) in the chart.

If there were a good way to embed numbers into charts themselves (getting rid of the tables altogether); this would be a better solution. We tried this, but the Google Chart API is limited, and we weren’t able to come up with a good solution.

9.8 A Fuller Use of YAML

As was discussed in section 2.1 on page 6, we have restricted our use of the YAML format. YAML is capable of more than we allow it to be. This restriction has simplified our work and is a cornerstone of our entire project. Among other things, it allows for us to perform a simple flattening process and easily map data into database tables. However, there might be some real advantages for loosening restrictions on our use of the YAML format.37 In particular, we’re thinking about result types that contain an array of varying length.

Consider the simple example mentioned in the conclusion of this document about using Mantevo Views to track workout data. Simply put, not every workout is the same. As an example, sometimes when I workout I’ll do four sets of bench press instead of three, and I want to collect data for each set; namely, repetitions and weight lifted. YAML data for a workout might look something like the following:

Listing 13: Workout - 3 sets

```yaml
---
Name: Workout
Version: 3
bench press:
  set:
    reps: 10
    weight: 275
  set:
    reps: 9
    weight: 315
  set:
    reps: 8
    weight: 275
---
```

Listing 14: Workout - 4 sets

```yaml
---
Name: Workout
Version: 4
bench press:
  set:
    reps: 12
    weight: 185
  set:
    reps: 11
---
```

37We realize this might mean restructuring a fair portion of the app.
But there's a better way to store this data in YAML, a way that doesn't require knowledge of the number of sets. The following YAML shows how YAML can represent an array of data:

```
Name: Workout
Version: 1.0
bench press:
  - set:
    reps: 12
    weight: 185
  - set:
    reps: 11
    weight: 225
```

Mantuvo Views doesn't know how to handle YAML like this, but this could change. The typical way of dealing with such a problem from a database standpoint would be to create an additional database table to aide the main result type table. In our workout example, we would have a result type table named Worksheet (or something similar), and we would also have a helper table named, perhaps, WorksheetSet. Worksheet would interact with WorksheetSet in a one to many relationship: each Worksheet tuple (a database term for a row) would have many WorksheetSet tuples associated with it while each WorksheetSet tuple would belong to a single Worksheet tuple.

Implementing this sort of a solution in Mantuvo Views seems possible. If Ruby were to convert the YAML of our last example into an object, it would use an array of bench press sets. Whenever Mantuvo Views encounters such an array structure, it could take note and create a new helper database table for each instance it comes across. Implementing this idea would considerably complicate the system. The benefits it would add would have to be weighed against the cost of complication. Unfortunately, it’s difficult to weigh costs and benefits without seeing an actual implementation. It might be the case that the added complexity is negligible (we highly doubt this). It might also be the case that the added functionality is negligible (we highly doubt this as well).

But this is really only half of the issue. Mantuvo Views not only stores data, but it displays data back to users in meaningful ways. Analyzing results of this new, more dynamic result type seems difficult and complex.
9.9 Deployment

Another area of further research should be focused on deployment. Jason Cross has helped us to deploy early versions of the app on CSBSJU servers, but the life of the app may continue outside of CSBSJU. It is essential that the app be easy to maintain. We used Capistrano, a deployment tool for Rails apps, and we envision using this tool in the future. We encourage whoever works on this project in the future to look at our deployment process with a critical eye and to use Jason Cross as a resource.

9.10 General Code Improvement

Although Ruby is an elegant and simple language, learning all of the Ruby idioms has proven to be a challenge. In Ruby, and especially in Rails, there always seems to be a better way of doing something. Being new to Ruby on Rails, we struggled to utilize some of the best coding practices. There are plenty of places to improve the code. Thankfully, we feel that we've done a good job of staying DRY, so the same code improvement should never need to be made in more than one place.

9.11 Site Administration

Admin capabilities are limited to adding new result types and editing existing result types. It seems that Admins could also deal with problem users and clean up unwanted results. Admins could be given the ability to view users and determine if they are causing problems by viewing their uploaded results. Admins could then send warning emails to users, delete “bad” results, and even delete users themselves.

9.12 Aesthetics

The app from a user's perspective is functional but not all that pretty. According to Andy Hunt, author of Pragmatic Thinking and Learning, “Several studies have conclusively shown that attractive user interfaces are easier to use than unattractive (or to use the scientific term, ugly) interfaces.” If for no other reason than increasing usability, the aesthetics of the app should be improved. Research needs to be conducted in cascading style sheets (CSS) and also JavaScript.

So far, no attempt has been made to test cross-browser compatibility. The app was developed and tested on Firefox 3.0. Care was taken to keep the app as simple as possible, and in this way, we hope we have avoided many cross-browser compatibility issues. However, further research is still needed in this area.

---

Thankfully, due to our achievements in code modularity and staying DRY, especially for result types, changes to the Mantevo Views’s views should be easy and non-repetitive.

### 9.13 AJAX and JavaScript

One nice feature of Ruby on Rails is its integration with AJAX (Asynchronous JavaScript and XML) and other JavaScript capabilities. We had been using AJAX as part of the paging system in HTML result tables, but we removed it because at the time it seemed to add unnecessary complexity to the code. The code which dealt with AJAX functionality was code that, at the time, was being generated for each individual result type. In an effort to generate as little code as possible, the AJAX code was removed. With our recent code refactoring, we’ve been pleased to realize that now we can add something like AJAX functionality in single place in the code (hooray for staying DRY), namely the result controller. With that said, we admit that although it would be an interesting feature, we question the overall benefit of adding AJAX to Mantevo Views. However, the benefits of AJAX or any other form of JavaScript may enhance Mantevo Views in ways we can’t currently foresee.

### 9.14 Rails Version

We developed in Rails 2.10, which was the most recent version of Rails when we started. Rails has been active since that time and is now at version 2.3. Rails will likely continue to come out with new versions. We have "frozen" the rails version at 2.10 in this Mantevo Views, but maybe a newer version would have some benefits. We don’t see there being a need to switch Rails versions in the near future.

---

*Freezing the version of Rails puts a copy of Rails itself into the /vendor/rails directory of a Rails app. When the app loads, it looks first in this directory, and if it finds a frozen version of Rails, it uses it.*
10 Conclusion

Although the webapp has yet to go live, the work of this thesis has been successful in certain areas. The standardization of the data gathering system has made the creation of a webapp like Mantevo Views possible. Some of the innovative ideas found in Mantevo Views, most notably the flexibility of the app in adding new result types, are of considerable worth in their own right. We have reason to believe that the features of Mantevo Views will increase the value and ease-of-use of Mantevo capabilities.

10.1 A General-Purpose Solution

We also have reason to believe that we have created a general-purpose solution, and that our work has possible applications beyond the initial scope of the project. Consider for a moment the way our system treats result types. All that’s unique to a result type is a name and version pair.\(^4\)

The other data of the result type could be anything at all. For example, I personally enjoy exercising. If I wanted to have a method of tracking my workouts, I could create a result type with a name “workout” and a version “1.0.” Before creating the new result type, I would have to decide what data I’d be interested in tracking. For example, how much time I spent stretching, how much time I spent running, and how much time I spent vainly flexing in the mirror. Our system would not only track this data for me, but it would provide me with some basic analysis tools. In a detail view, I could create pie charts to analyze how much time I spent in each of the various portions of my workouts. Using a worksheet, I could examine multiple workouts and create radar charts to quickly analyze the differences between them.

But, this is just one example of how our system can be used as a general-purpose data storage and analysis tool; we can think of others: a teacher could use our system to track and analyze student data, such as grades; a basketball league could use our system to track and analyze important team statistics, such as shooting percentage; and, a clown at a Paradiso restaurant could use our system to track whatever strange sorts of data he would be interested in, such as the number of balloon wiener dogs he made, the number of customers who fell for his fake camera that really squirts water, and the number of kids he made laugh. While not all of these ideas are practical, the point remains that our system is a surprisingly general-purpose solution to the problem of data storage and analysis.

\(^4\) Considering Mantevo Views as a general-purpose data storage and analysis tool, our choice of the term result is a little inappropriate. We chose to use the term result for Mantevo Views because Mantevo users gather data when running miniapplications, which can be thought of as tests that produce results. When thinking of our system in this general sense, it probably makes more sense to use a phrase like data-group type instead of result type.
10.2 Was Ruby on Rails a Good Choice?

We believe the choosing to develop our webapp with Ruby on Rails was overall a great choice. With that said, it’s difficult for us to compare it to our other options, such as PHP, simply because our experience is limited to working with Ruby on Rails. The work of this thesis has given us a solid understanding of the Ruby programming language and of the Rails framework. In short, we like what we’ve found. We agree with Ruby on Rails’s claims of power and simplicity, though we recognize becoming a proficient Rails developer is far from a trivial task.

The amount of online support for learning Ruby on Rails is fantastic, especially for an open-source framework. Blogs, screencasts, and message boards abound with information on solving common Rails problems. In this way, the Ruby on Rails community is exceptionally active. From what we can tell, other web development frameworks fall short of the level of support Rails has attained.

Other concerns about Rails, such as security concerns, have largely fallen away. The concern isn’t whether Ruby on Rails itself is secure, but whether we can develop our app in a secure fashion.

Finally, Ruby is an extremely flexible programming language. What it lacks in speed, it makes up for in ease-of-use. Some of the “tricks” we employed with dynamically handling result types were made possible by Ruby’s interpreted nature. We believe that implementing these tricks in other languages would have been considerably more difficult, if even possible at all.

10.3 Lessons Learned

Real world problems are open-ended, which is a very different thing than the typically closed-ended problems encountered in academia. It’s often tempting when working with an open-ended problem to discover a solution and latch onto it, but true innovation comes from constantly searching for new and improved solutions. Mike Heroux’s open-mindedness and insistence in always pursuing new avenues, even if we thought we had arrived at an adequate solution, was the foundation for all of our major successes. As we look ahead to what may become of this project in the future, we try to maintain a similar attitude toward problem solving and development. The work of this thesis has created a decently large software system, but this system is not fixed. Believing it to be fixed curtails innovation.

If we were to start over from the beginning, we might place a larger emphasis on refactoring and good coding practices from the outset. Code refactoring is not something that should happen at the end of a development process; it should be taking place throughout. Refactoring isn’t so much about beautifying the code as it is about simplifying the code and freeing oneself up to tackle problems in a new way. It was only after we began to utilize inheritance in result controllers that we truly realized the power we had created with the ResultType table. All of a sudden Mantevo Views became a whole lot more dynamic, simple, and flexible.
On a personal note, I've learned the value of group work. Interacting with other people while problem solving can be a great source of inspiration and enjoyment. The work of this thesis has been carried out far too often in isolation. Also, from a productivity point of view, I used to believe that I worked best alone, and that adding another person to my one-man team would not even come close doubling productivity. Now, I'm not so sure. With easy, closed-ended problems, I may have been correct, but with difficult, open-ended problems, teamwork is essential.

10.4 The Future of the Project

The work of this thesis has been a process of continual improvement. As I prepare to walk away from the project, my understanding of Ruby, Rails, and the system we have created is at its height. Over the past few months, after all of the major design considerations were out of the way, we have been busy improving the code and the documentation making it as easy as possible for someone to continue the work of this thesis in the future. We believe that Mantevo Views will continue to improve, and that it will ultimately serve its purpose storing and analyzing data for the Mantevo community.
A Glossary

each version of each miniapplication  this phrase alludes to the structure of the database where there is one table for each version of each miniapplication. In other words, each version of each miniapplication is it’s own result type. For example, if fictitious miniapplication SJU had two versions, 1.0 and 2.0, there would be a result table for each.

Mantevo Views the name of the Ruby on Rails webapp.

miniapplication the name used for miniapplications, application proxies, and minidrivers. No distinction is made between the three in this document.

result the output of a miniapplication. Miniapplications can be thought of as tests that produce results. Results are in the YAML format, and once uploaded to Mantevo Views, they are mapped into result tables. Each result belongs to a result type, which is determined by a result’s name-version combination.

result model any of the various Ruby model classes (see code in /app/models) for wrapping a result table.

result table a table in Mantevo Views for storing results of a given type.

result type the classification of a result. There is one result type for each version of a miniapplication. Result types are determined the name and version of results.
B Adding a New Result Type

Creating a new result type is only done in development mode. Rails has essentially three different modes of operation: development, testing, and production. For anyone continuing research on this project, it is extremely important to understand the differences. In development mode, an admin can login and upload a YAML result for a new result type. The following is an overview of the process of adding a result type:

1. Check out a copy of the project from the svn repository.
2. Run the rake task to set up the database.
3. Start the development server on localhost.
4. Log in as an admin.
5. Upload the new YAML result file.
6. Decide what data should be displayed in a result HTML table for this new result type.
7. Provide alternate names to replace ugly column names.
8. Click 'finish' and run the result rake task.
9. Commit changes back to the repository.
10. Run Capistrano commands to bring changes to the deployed application.
11. Run the rake task to set up the deployment database.

More details will be provided in the README file in the doc directory of the app.
C Continuing Research

Mantevo Views is a Ruby on Rails webapp. This document is intended to help someone pick up the project where I left off. It is rather short because most of the information pertinent to continuing research is found in the code documentation. We have done our best to document the code in such a way that helps a newcomer take over the project. Code documentation can be found the the /doc/app directory of the app. Ruby Rdoc documentation is the Rails default for documentation. It creates HTML files and is very similar to Javadoc. To generate documentation for Mantevo Views, navigate to the root directory of the app and run “rake doc:app.” This will generate new documentation or update the old documentation in the /doc/app directory.

Private methods and comments within methods are not included in the Rdoc documentation. Because of this, going strait to the source might often be the easiest way of understanding Mantevo Views. Plus, Ruby code is generally very easy to read.
C.1 Installing Rails

I recommend developing Mantevo Views in a Linux environment: it has worked well for me and it adheres to our unwritten policy of only using open source tools. My guide was *Agile Web Development with Rails*, 3rd edition. This book gives instructions for installing Rails on Windows, Mac OS X, and Linux. The following is a short description of the process I ran through to get Rails up and running on my system (Ubuntu 8.10).

- First, I ran these commands:
  
  ```bash
  sudo aptitude update
  sudo aptitude install build-essential libopenssl-ruby
  sudo aptitude install ruby1.8-dev libsqlite3-dev
  ```

- Next, I updated Rubygems:

  ```bash
  sudo gem install rubygems-update
  cd /var/lib/gems/1.8/gems/rubygems-update-*
  sudo ruby setup.rb
  ```

Note: Thomas says, “There are many different ways to upgrade Rubygems, unfortunately based on which version of gems you have installed and what distribution you are running, not all of the ways work. Be persistent.” If the method of updating Rubygems that I used above doesn’t work for you, try one of the following:

- Using the gem update system:

  ```bash
  sudo gem update --system
  ```

- Using the gem designed to update troublesome systems:

  ```bash
  sudo gem install rubygems-update
  sudo update_rubygems
  ```

- Finally, installing from source:

  ```bash
  wget http://rubyforge.org/frs/download.php/45905/rubygems-1.3.1.tgz
  tar xzf rubygems-1.3.1.tgz
  cd rubygems-1.3.1
  sudo ruby setup.rb
  ```

Because I have “frozen” rails version 2.1.0 into Mantevo Views (the rails source actually exists in the /vendor/rails directory of Mantevo Views), this next step isn’t actually necessary. When launching the app running ‘script/server’, the script looks first to see if rails has been frozen (i.e. if it exists in the vendor directory), and if it finds nothing, only then will it look for a system-wide installation of Rails.
sudo gem install rails (this is not necessary to use Mantevo Views because the gem is frozen, but it doesn’t hurt anything)

- Finally, run one last command:

  
  sudo gem install sqlite3-ruby

You may also need to add /var/lib/gems/1.8/bin to your PATH environment variable, but I did not in Ubuntu 8.10.

That’s it! Your system should be ready to go. Navigate to Mantevo Views and run ‘script/about’. My output when running this command is the following:

  About your application’s environment
  Ruby version 1.8.7 (x86_64-linux)
  RubyGems version 1.3.1
  Rails version 2.1.0
  Active Record version 2.1.0
  Action Pack version 2.1.0
  Active Resource version 2.1.0
  Action Mailer version 2.1.0
  Active Support version 2.1.0
  Edge Rails revision unknown
  Application root /home/cam/Mantevo_Results
  Environment development
  Database adapter sqlite3
  Database schema version 20090202121855

You should have similar output. If you do, your installation should be good to go.
C.2 Learning Ruby on Rails

If you plan on making real changes to the app, you’ll need to understand Ruby on Rails. With that said, certain aesthetic changes can be made to the app without much difficulty. For example, changes to the CSS (cascading style sheets) may be made without any understanding of Rails whatsoever. For more about making aesthetic changes to the app, please see section.

Learning Ruby on Rails can be enjoyable, especially if you’re a CS student used to programming in Java. If I had to sum up the Ruby programming language in a single word, I would call it elegant. If I had to sum up the Rails framework in a single word, I would call it powerful. Ruby on Rails is surprisingly intuitive and natural. If you end up spending significant time with Mantevo Views, I hope you will come to enjoy Ruby on Rails as I have.

C.2.1 Learning Ruby

If most of your programming experience is in Java, I recommend starting out by reading “To Ruby From Java,” a short web page highlighting the differences between the two languages. After that, I would explore the many other free resources available online (check out the main Ruby site at www.ruby-lang.org/en/).

Depending on how much you’ll be working with Mantevo Views, it might be worth purchasing “Programming Ruby” by Dave Thomas, which I personally found exceptionally helpful. The book is currently in its third edition, but, for what it’s worth, the first edition is actually available for free online at www.rubycentral.com/book/. Also, professor Jim Schnef who taught a class at St. John’s on Ruby on Rails a couple years ago (I wasn’t in the class) owns a hard copy of the second edition.

From personal experience learning the language, I found the Interactive Ruby Shell (IRB) to be one of the best tools for learning the language. IRB allows you to quickly write code and quickly test it; IRB is an experimentation playground. If you don’t know what it is, simply type IRB from the command line (you may need to install Ruby and IRB first). Check out tryruby.hobix.com to play around with IRB online.

C.2.2 Learning Rails

As with Ruby, there are plenty of good, free online resources for learning Rails. I recommend starting at rubyonRails.org and going from there. You might also consider a non-free resource called “Agile Web Development with Rails” by Dave Thomas. I want to offer a word of caution about using anything other than the 3rd edition of this book. Rails has changed quite rapidly. Mantevo Views uses version 2.10 of Rails. The 2nd edition of the book was written before Rails 2.0,
and as I learned the hard way, many of the examples in the second edition don’t work in Rails 2.0 or greater.

Look into using 'script/console' as a learning tool. Once you have rails installed on your system, run 'script/console' from the root directory of a rails app and you have access to an IRB session with the rails environment loaded and ready for you to play around with.
C.3 Text Editor

When developing in Rails, one doesn’t need a complicated IDE. Ruby is different from Java or C++ in this regard. In essence, Ruby’s simplicity negates the need to develop in anything much more advanced than a simple text editor. While Mac users love textmate for writing Rails apps, I have enjoyed using gedit, the default text editor in Ubuntu. There are other text editors available for developing Rails on Linux, but of those I tried, I like gedit the best.

Gedit needs some modification to make it ready for rails. Fortunately, this modification is extremely simply if you go to the right place. I recommend going to http://github.com/mig/gedit-rails/tree/master. This download describes itself as “a collection of scripts and configuration files that make gedit a more rails-friendly IDE.” After downloading, it’s a simple matter of extracting the compressed folder and running the installation script. After this, I suggest editing your preferences within gedit to customize your text editor exactly the way you want it. For example, in the editor tab of the gedit preferences, I recommend setting gedit to insert spaces instead of tabs, which is useful when writing ruby code or editing YAML:

Here is a screenshot of what my gedit looked like after my modifications:
require 'yaml'

# This controller is responsible for all code generation
# The public methods correspond to the different steps a user takes through the
# upload process.
# The private methods correspond to the different files to be generated.

class AdminUploadController < ApplicationController
  layout 'admin'
  before_filter :authorize_admin

  def index
    session[:upload] = Hash.new
  end

  # This method checks the validity of the uploaded file. If the file passes the
  # tests, a new entry is made in the upload database, and a user can continue the
  # upload process.
  def submit
    # If this method was called as a result of a file upload...
    if request.post?
      submission = params[:file]
      @submission = submission.read if submission.respond_to? :read

      # If the file was readable
      if @submission
        # If the file contained proper YAML
        if @yaml = YAML::load(@submission)
          if app = @yaml.delete('Mini:Application Name')
            if version = @yaml.delete('Mini:Application Version').to_s
              # since the @name variable is used to name classes later on, it
              # must conform to some standards. First, it cannot have numbers
              # immediately following an underscore. Second, it cannot have
              # a decimal. I'm looking into a dash...
              @name = @yaml.delete 'Mini:Application Version'
            end
          end
        end
      end
    end
  end
end
D Source Code

If you would like a digital copy of the source, please contact the author at camaration@gmail.com.
D.1 Data Gathering Tools

The code of this section was taken from the trunk of the Mantevo code repository on Mar 27, 2009. From the root directory of Mantevo, the following files can be found in packages/common/. Mike Heroux has added Doxygen style comments to the code.

D.1.1 YAML_Doc.cpp

Listing 16: YAML Doc

```cpp
#include <vector>
#include <iostream>
#include <fstream>
#include "YAML_Doc.hpp"
using namespace std;

// set the microapp_name and version which will become part of the YAML doc.
YAML_Doc::YAML_Doc(string miniAppName, string miniAppVersion, std::string destinationDirectory, std::string destinationFileName)
    : this->miniAppName = miniAppName;
    this->miniAppVersion = miniAppVersion;

// inherits the destructor from YAML_Element
YAML_Doc::~YAML_Doc()
{
}

// generates YAML from the elements of the document and saves it
// to a file
string YAML_Doc::generateYAML()
{
    string yaml = "---
";
    yaml = yaml + "Mini-Application Name: " + miniAppName + "\n";
    yaml = yaml + "Mini-Application Version: " + miniAppVersion + "\n";
    for (int i = 0; i < children.size(); i++)
    {
        yaml = yaml + children[i]->printYAML(" ");
    }

time_t rawtime;
    tm * tmptm;
    time(&rawtime);
    tmptm = localtime(&rawtime);
    char sdate[25];
    strftime(sdate, "%04d:%02d-%02d:%02d:%02d", tmptm, tm_mon, tm_mday, tm_hour, tm_min, tm_sec);

    string filename;
    if (destinationFileName =="
        filename = miniAppName + "-" + miniAppVersion + "_
        else
          filename = destinationFileName;

    filename = filename + string(sdate) + ".yaml";
    if (destinationDirectory!="")
    {
```
```cpp
    string mkdir = "mkdir " + destinationDirectory;
    system(mkdir.c_str());
    filename = destinationDirectory + "/" + destinationFileName;
  }
  else
    filename = "./" + filename;
  ofstream myfile;
  myfile.open(filename.c_str());
  myfile << yaml;
  myfile.close();
  return yaml;
```

D.1.2 YAML_Doc.hpp

Listing 17: YAML Doc header

```cpp
// @HEADER
// */***********************************************************************/
// Mantle: A collection of mini-applications for HPC
// Copyright (2008) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
// Questions? Contact Michael A. Heroux (maherou@sandia.gov)
//
// ***********************************************************************/
// @HEADER
// Changelog
//
// Version 0.1
// - Initial version.
//
// #ifndef YAML_DOC_H
#define YAML_DOC_H
```
```cpp
#include <string>
#include <vector>
#include "YAML_Element.hpp"

// The Mantetov YAML_Doc class for the uniform collecting and reporting of performance data for mini-applications

The YAML_Doc class works in conjunction with the YAML_Element class to facilitate easy collecting and reporting of YAML-formatted data that can be then registered with the Mantetov results collection website.

code

// EXAMPLE CODE FOR GENERATING YAML

YAML_Doc doc("hpccg","1.0");
doc.add("final_residual",1.4523e-13);
doc.add("time","4.893");

// note: the following line will remove the data (4.890) associated with "time"
doc.get("time")->add("total",4.243);

// note: the following line will likewise remove the data (1.243) associated with "time"

doc.get("time")->get("total")->add("time",2.457);
doc.get("time")->get("total")->add("flops",4.88e5);
doc.get("time")->add("ddot",1.243);
doc.get("time")->add("sparselmv",");
doc.get("time")->get("sparselmv")->add("time",0.3445);
doc.get("time")->get("sparselmv")->add("overhead",");
doc.get("time")->get("sparselmv")->get("overhead")->add("time",0.0123);
doc.get("time")->get("sparselmv")->get("overhead")->add("percentage",0.034);

cout << doc.generateYAML() << endl;
return 0;

//endcode

Below is the output generated by the above code:

verbatim

final_residual: 1.4523e-13
time:
  total:
    time: 2.457
flops: 4.88e5
ddot: 1.243
sparselmv:
  time: 0.3445
overhead:
  time: 0.0123
percentage: 0.034

endverbatim
```
D.1.3 YAML_Element.cpp

Listing 18: YAML Element

```cpp
#include <vector>
#include <iostream>
#include <fstream>
#include <sstream>
#include "YAML_Element.hpp"

using namespace std;

YAML_Element::YAML_Element(string key, string value) {
    this->key = key;
}
```
```cpp
this->value = value;

YAML_Element::YAML_Element() {
    int i;
    for (i = 0; i < children.size(); i++) {
        delete children[i];
    }
    children.clear();
}

*/
/* Add an element to the vector
 * QUESTION: if an element is not added because the key already exists, 
 * will this lead to memory leakage?
*/
YAML_Element* YAML_Element::add(string key, double value) {
    this->value = "";
    string converted_value = convert_double_to_string(value);
    YAML_Element* element = new YAML_Element(key, converted_value);
    children.push_back(element);
    return element;
}

YAML_Element* YAML_Element::add(string key, int value) {
    this->value = "";
    string converted_value = convert_int_to_string(value);
    YAML_Element* element = new YAML_Element(key, converted_value);
    children.push_back(element);
    return element;
}

YAML_Element* YAML_Element::add(string key, size_t value) {
    this->value = "";
    string converted_value = convert_size_t_to_string(value);
    YAML_Element* element = new YAML_Element(key, converted_value);
    children.push_back(element);
    return element;
}

YAML_Element* YAML_Element::add(string key, string value) {
    this->value = "";
    YAML_Element* element = new YAML_Element(key, value);
    children.push_back(element);
    return element;
}

*/
/* returns pointer to the YAML_Element for the given key. 
 * I, cam, believe an exception should be thrown if there is no 
 * element in the vector for the specified key 
*/
YAML_Element* YAML_Element::get(string key) {
    int i;
    for (i = 0; i < children.size(); i++) {
        if (children[i]->getKey() == key) {
            return children[i];
        }
    }
    return NULL;
}
```
D.1.4 YAML_Elemen t.hpp

Listing 19: YAML Element header
// ** Header **

// Changelog

// - Version 0.1
// - Initial version.

#include <string>
#include <vector>

/*! The Mantevo YAML_Element class for registering key-value pairs of performance data */

class YAML_Element {
public:

  // Default constructor.
  YAML_Element() { key=""; value=""; }

  // Construct with known key-value pair
  YAML_Element (std::string key, std::string value);

  // Destructor
  ~YAML_Element () ;

  // Key accessor method
  std::string getKey() {return key; }

  // Add a child element to an element list associated with this element, value of type double
  YAML_Element* add(std::string key, double value);

  // Add a child element to an element list associated with this element, value of type int
  YAML_Element* add(std::string key, int value);

  // Add a child element to an element list associated with this element, value of type size_t
  YAML_Element* add(std::string key, size_t value);

  // Get the element in the list with the given key
  YAML_Element* get(std::string key);
D.1.5 gather_yaml.pl

This script is obsolete. See section and section.

Listing 20: Gather YAML

```perl
#!/usr/bin/perl
use strict;
use warnings;

my $all_yaml;

run();

#------------------------------
# run()
#
# generate yaml and write it to a file
# - args: none
# - returns: none
#
sub run {
  get_sys_info();
  my $dir = "../";
  opendir(DIR, $dir) or die "can't open $dir: $!
"
  my @files = readdir(DIR);
  closedir(DIR);
  foreach my $file (@files){
    if (-T "$dir/$file"){
      get_result_info("$dir/$file");
    }
  }
  print $all_yaml;
  my $filePath = $dir . "upload/" . "mantevo_results.yaml";
  open(my $out, "">", $filePath) or die "Can't open $filePath: $!
"
  print $out $all_yaml;
} # run
```
sub get_sys_info {
    my $hostName = "";
    my $dnsName = "";
    my $ipAddress = "";
    my $operatingSystem = "";
    my $kernelName = "";
    my $kernelRelease = "";
    my $kernelVersion = "";
    my $processor = "";
    my $machineHardware = "";
    my $hardwarePlatform = "";
    my $badCmd = 0;
    my $outString = "";
    my $time = "";
    $badCmd = system ('hostname -s > /dev/null 2>&1');
    if (!$badCmd) {
        chomp($hostName='hostname -s');
    } else {
        $hostName=getOptionalOutput("uname -n");
    }
    $dnsName=getOptionalOutput("hostname -d");
    $ipAddress=getOptionalOutput("hostname -i");
    $operatingSystem=getOptionalOutput("uname -o");
    $kernelName=getOptionalOutput("uname -s");
    $kernelRelease=getOptionalOutput("uname -r");
    $kernelVersion=getOptionalOutput("uname -v");
    $processor=getOptionalOutput("uname -p");
    $machineHardware=getOptionalOutput("uname -m");
    $hardwarePlatform=getOptionalOutput("uname -i");
    $time = time();
    $all_yaml = "---
    description: $time
    hostName: $hostName
dnsName: $dnsName
ipAddress: $ipAddress
operatingSystem: $operatingSystem
kernelName: $kernelName
kernelRelease: $kernelRelease
kernelVersion: $kernelVersion
processor: $processor
machineHardware: $machineHardware
hardwarePlatform: $hardwarePlatform"
};

# get_result_info()

# appends the yaml from a result file to the "all_yaml" string

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
sub get_result_info {
    my $current_file = $_[0];
    local $/=undef;
    open FILE, $current_file or die "Couldn't open $current_file: $!";
    my $yaml_string = <FILE>;
    close FILE;
    $all_yaml = "$all_yaml---SEPARATOR---\n\n$yaml_string\n";
} # get_result_info() 

# getOptionalOutput()
# Runs a system command and gets its output if it succeeds then returns
# its output. The program suppress output to stderr of the program does
# not run correctly.
# - args: the shell command
# - returns: the optional output string
#
sub getOptionalOutput {
    my $cmd_in = shift;
    my $cmd = "$cmd_in > /dev/null 2>&1";
    my $returnVal = system($cmd);
    chomp($outputStr = `$cmd_in`) if($returnVal == 0);
    return $outputStr;
} # getOptionalOutput()
### E Mantevo Medley Code

This section contains the source code for Mantevo Medley. A much better way to view code and comments is through Ruby's RDoc utility. Rails has a rake task for generating RDoc documentation (this is similar to Javadoc).

#### E.1 Controllers

The following files can be found in `/app/controllers/`

```
#huh

class AdminLoginController < ApplicationController
  layout 'admins'
  #index could have been named 'login', but being that it is in the AdminLogin
  #controller it seemed redundant.
  def index
    #don't allow user and admin to be logged in at the same time
    session[:user_id] = nil

    #if the user is trying to log in...
    if request.post?
      #the authenticate method is a special method written in the Admin model
      admin = Admin.authenticate(params[:name], params[:password])

      #if the authentication passed...
      if admin
        session[:admin_id] = admin.id

        #if the user was redirected to the login, the url the user attempted
        #to access was stored in the session (see authenticate_admin in application.rb)
        #and will be used to send the user back to that url.
        uri = session[:original_uri]
        session[:original_uri] = nil
        redirect_to(uri || { :controller => :admins, :action => "index" })
      else
        flash.now[:notice] = "Invalid Username/Password combination"
      end
    end
  end

  def logout
    session[:admin_id] = nil
    flash[:notice] = "You've been logged out"
    redirect_to :controller => :admin_login, :action => :index
  end
end
```

<table>
<thead>
<tr>
<th>Listing 21: Admin Login Controller</th>
</tr>
</thead>
</table>

```
class AdminsController < ApplicationController
  before_filter :authorize_admin
```
layout 'admins'

# GET /admins
# GET /admins.xml
def index
  @admins = Admin.find(:all, :order => :name)
  respond_to do |format|
    format.html # index.html.erb
    format.xml { render :xml => @admins }
  end
end

# GET /admins/new
# GET /admins/new.xml
def new
  @admin = Admin.new
  respond_to do |format|
    format.html # new.html.erb
    format.xml { render :xml => @admin }
  end
end

# GET /admins/1/edit
def edit
  @admin = Admin.find(params[:id])
end

# POST /admins
# POST /admins.xml
def create
  respond_to do |format|
    if @admin.save
      flash[:notice] = 'Admin was successfully created.'
      format.html { redirect_to(@admin) }
      format.xml { render :xml => @admin, :status => :created, :location => @admin }
    else
      format.html { render :action => "new" }
      format.xml { render :xml => @admin.errors, :status => :unprocessable_entity }
    end
  end
end

# PUT /admins/1
# PUT /admins/1.xml
def update
  @admin = Admin.find(params[:id])
  respond_to do |format|
    if @admin.update_attributes(params[:admin])
      flash[:notice] = 'Admin was successfully updated.'
      format.html { redirect_to(admins_path) }
      format.xml { }
require 'yaml'

# The AdminUploadController gathers all necessary information
# for adding a new maneto result. Only admins can step through this process.
#
# Methods correspond to the different steps a user takes through the
# upload process.
class AdminUploadController < ApplicationController
  layout 'admins'
  before_filter :authorize_admin
  before_filter :admin_has_upload, :except => [:index, :submit]
  before_filter :set_app_and_ver, :except => [:index, :submit]

  # The filters
  protected

  # This method is used as a before filter and serves to redirect a user
  # back to the index if he or she does not possess an upload. A user gains
  # an upload by submitting a YAML file.
  def admin_has_upload
    unless @admin.upload
      redirect_with_msg nil
    end
  end

  end

  def set_app_and_ver

Listing 23: Admin Upload Controller
upload = Admin.upload
@app = upload.app
@version = upload.version
end

def index
end

# Tests the validity of the uploaded file. If the file passes the
tests, a new entry is made in the upload database, and the user can continue
# with the upload process.
def submit

  # if this method was called as a result of a file upload...
  if request.post?
    submission = params[:file]
    submission = submission.read if submission.respond_to?(:read)
  # if the file was readable...
  if submission
    # if the file contained proper YAML...
    if yaml = YAML::load(submission)
      # if the YAML contained a 'Mini-Application Name' and a
      # 'Mini-Application Version'...
      app = yaml.delete('Mini-Application Name')
      version = yaml.delete('Mini-Application Version').to_s
      if app
        load "#{RAILS_ROOT}/lib/result_helper.rb"
        name = ResultHelper.generate_name app, version
        unless MiniApp.find_by_name name
          load "#{RAILS_ROOT}/lib/hash_methods.rb"
          flat_yaml = yaml.flatten
          flat_yaml = flat_yaml.cleanup_keys
          alternate_names = flat_yaml.generate_alternate_names
          unless
            # Create flat YAML and alternate names
            load "#{RAILS_ROOT}/lib/hash_methods.rb"
            flat_yaml = yaml.flatten
            flat_yaml = flat_yaml.cleanup_keys
            alternate_names = flat_yaml.generate_alternate_names
            # Instead of storing upload information in the session, data is
            # stored in the Uploads table. Note that the YAML format is used
            # for hashes and arrays (hence the constant '.to_yaml' and
            # YAML::load statements throughout the code.
            begin
              # Each admin has one and only one upload. If an upload already
              # exists, destroy it.
              @admin.upload.destroy if @admin.upload
              upload = Upload.new
              upload.name = name
              upload.app = app
            end
          end
        end
      end
    end
  end
end
upload.version = version
upload.flat_yaml = flat_yaml.to_yaml
upload.alternate_names = alternate_names.to_yaml
upload.columns = Array.new
upload.admin_id = @admin.id
upload.save

# if nothing went wrong...
redirect_to :action => :edit_column_names
rescue
  redirect_with_msg "Failure saving upload information"
end

# Something went wrong somewhere along the line, redirect with an appropriate
# message.
else
  redirect_with_msg "This result type already exists"
end

# else
  redirect_with_msg "Mini-Application Version not given"
end
else
  redirect_with_msg "Mini-Application Name not given"
end
else
  redirect_with_msg "Improper YAML format"
end
else
  redirect_with_msg "Unable to read file"
end
else
  redirect_with_msg "Unable to read file"
end
end

# This method allows the user to alter the column names of the table because
# many of the generated names names get really long and ugly, especially
# when the submitted YAML has a lot of heirarchy.
#
# Note: these alternate names don't change the real
# column names in the database. Instead, they simply
# change the way names are displayed back to the user through the browser.
def edit_column_names
  upload = @admin.upload
  @alternate_names = YAML::load(upload.alternate_names)

  # if the user is trying to edit the names...
  if request.post?
    @alternate_names.each_key {{key | @alternate_names[key] = params[key] if
    params[key] }
    upload.alternate_names = @alternate_names.to_yaml
    upload.save
  end
end
# Allows the user to select what he or she views to be the most important
# columns for display in an HTML mantovu results table.
def column_selection

  upload = @admin.upload
  @columns = YAML::load(upload.columns)
  @flat_yaml = YAML::load(upload.flat_yaml)
  @alternate_names = YAML::load(upload.alternate_names)

  # if the user is trying to add columns...
  if request.post?
    @alternate_names.each_key do |key|
      @columns.delete key if @columns.include? key
      @columns << key
    end
  end
  upload.columns = @columns.to_yaml
  upload.save
end

# Remove a column (from an HTML table...). Always redirect back
# to column selection.
def remove_column

  upload = @admin.upload
  @columns = YAML::load(upload.columns)
  @columns.delete(params[:id])
  upload.columns = @columns.to_yaml
  upload.save
  redirect_to :action => :column_selection
end

# Allow a user to look over the upload before finishing and remind the
# user that alternate names and columns are changeable later.
def review

  upload = @admin.upload
  @flat_yaml = YAML::load(upload.flat_yaml)
  @alternate_names = YAML::load(upload.alternate_names)
  @columns = YAML::load(upload.columns)
end

# Convert the upload entirely to YAML and save the file. The view
# instructs the user what to do from here.
def finish_upload

  upload = @admin.upload
  @filename = upload.name
  File.open("#{RAILS_ROOT}/results/#{@filename}", 'w+'){ |f| f.write(upload.to_yaml) }

end
Listing 24: Application

```ruby
# Filters added to this controller apply to all controllers in the application.
# Likewise, all the methods added will be available for all controllers.
class ApplicationController < ActionController::Base
  helper :all # include all helpers, all the time

  # See ActionController::RequestForgeryProtection for details
  # Uncomment the :secret if you're not using the cookie session store
  protect_from_forgery :secret => '4b4c010c89e0546e804abef53b74ade'

  before_filter :set_page_title
  protected

  # Each controller can choose how it wants to set the page title, but the default
  # will be 'Mantevo'
  def set_page_title
    @page_title = 'Mantevo'
  end

  # Use this method as a before filter when you need to make sure the current
  # user is logged-in. It redirects non-logged-in users to the LoginController.
  def authorize_user
    @user = User.find_by_id(session[:user_id])
    unless @user
      session[:original_uri] = request.request_uri
      flash[:notice] = "Please log in"
      redirect_to :controller => :login, :action => :index
    end
  end

  # Use as a before filter in the admin controllers.
  def authorize_admin
    @admin = Admin.find_by_id(session[:admin_id])
    unless @admin
      session[:original_uri] = request.request_uri
      flash[:notice] = "Please log in"
      redirect_to :controller => :admin_login, :action => :index
    end
  end
end
```

74
class LoginController < ApplicationController
layout 'main'

# index could have been named 'login', but being that it is in the AdminLogin # controller it seemed redundant.
def index
  # don't allow user and admin to be logged in at the same time
  session [:admin_id] = nil
  
  # if the user is trying to log in...
  if request.post?
    login
  end
end

def logout
  session [:user_id] = nil
  flash [:notice] = "You've been logged out"
  redirect_to :controller=> :login, :action => :index
end

# Allows a new user to be created.
# Some testing needs to be done on this method to make sure that people
# can't register improperly.
def register
  if request.post?
    # create the user
    @user = User.new (:email => params [:email],
      :password => params [:password],
      :password_confirmation => params [:password_confirmation])
    if @user.save
      flash [:notice] = "User #{@user.email} was successfully created."
    else
      @user = nil
      flash.now [:notice] = "Registration failure. Please try again."
    end
    if @user
      login
    else
      @user = User.new
    end
  end
private
  def login
    user = User.authenticate (params [:email], params [:password])
    # if the authentication passed...
    if user
session[:user_id] = user.id

# SMART REDIRECT: if the user was redirected to the login, the url the user attempted
# to access was stored in the session (see authenticate_admin in application.rb)
# and will be used to send the user back to that url.
uri = session[:original_uri]
if uri
  redirect_to(uri, { controller: :user, action: "index" })
else
  flash.now[:notice] = "Invalid Email/Password combination"
end

def set_page_title
  @page_title = "Login | #{self.action_name.titleize}"
end
before_filter :worksheet_chart_setup, :only => [:create_worksheet_charts, :remove_worksheet_chart, :edit_worksheet_chart, :worksheet_remove_column]

protected

# Get user from the database and assign it to @user
def authorize
  @user = User.find_by_id(session[:user_id])
end

# Redirect if @user is nil
def no_user_redirect
  redirect_to[:controller => :login, :action => :index] unless @user
end

# Set @app_and_ver based on the controller name
def set_app_and_ver
  @app_and_ver = self.controller_name
end

# Because result controllers inherit public methods from ResultController, these
# methods are available in a URL such as /result/create_worksheet_charts, which
# is not what we want. If a user manually enters such a URL, redirect to
# /result/home
def check_controller_name
  if (self.controller_name == "result" && self.action_name != "home") ||
    (self.controller_name != "result" && self.action_name == "home")
    redirect_to :controller => :result, :action => :home
  end
end

# Set the page title in the 'setup_columns_and_alternate_names' before filter
# instead.
def set_page_title
end

# Used to make creating the HTML tables a dynamic process. Also used when making
# charts
def setup_columns_and_alternate_names
  if self.controller_name != "result"
    @app = MiniApp.find_by_name(@app_and_ver)
    if @app
      @alternate_names = YAML::load(@app.alternate_names) if @app
      @columns = YAML::load(@app.columns) if @app
      @page_title = "#{@app.app} #{@app.version} | #{self.action_name.titleize}"
    else
      flash[:notice] = "There is no entry in MiniApp for result type you have
      requested"
      redirect_to :controller => :result, :action => :home
    end
  end
end

# Get the detail charts for a user from DetailChart and assign them to
# @detail_charts.
def detail_chart_setup
if @user

#the next three lines guarantee that @detail_charts is an array
@detail_charts = @user.detail_charts.find :all, :conditions => ['name == ?', @app_and_ver]
@detail_charts = [] unless @detail_charts
@detail_charts = [ @detail_charts ] unless @detail_charts.class == Array
@flat_yaml = YAML::load( @app.flat_yaml)
end
end

# Get the worksheet charts for a user from WorksheetChart and assign them to @worksheet_charts.
def worksheet_chart_setup
  if @user
    #the next three lines guarantee that @worksheet_charts is an array
    @worksheet_charts = @user.worksheet_charts.find :all, :conditions => ['name == ?', @app_and_ver]
    @worksheet_charts = [] unless @worksheet_charts
    @worksheet_charts = [ @worksheet_charts ] unless @worksheet_charts.class == Array
    @flat_yaml = YAML::load( @app.flat_yaml)
  end
end

# Nothing happens in the home method yet. Maybe it should be a simple
# description about the project and the app.
def home
end

# Prepare results for the view. Note: if the code is confusing, @app_and_ver
# is set in a before_filter. It is turned into a constant which allows it
# to function as a model class from which to draw results. The paginate
# method makes is from the will_paginate plugin (see /vendor/plugins)
def index
  @results = @app_and_ver.camelize.constantize.paginate :page=>params[:page],
            :per_page=>RESULTS_PER_PAGE,
            :order => params[:sort]
  respond_to_results # For xml purposes
end

# See index. Instead of getting all results, only get the ones that belong to
# the logged-in user.
def user
  @results = @user.send( @app_and_ver.pluralize ).paginate :page=>params[:page],
             :per_page=>RESULTS_PER_PAGE,
             :order => params[:sort]
  respond_to_results # For xml purposes
end

private

# If the user wants xml, give it.
def respond_to_results
  respond_to do |format|
    format.html { render: partial => 'result/results', :layout => 'main'
  format.xml do

```ruby
results = @results.collect do |result|
  result = YAML::load(result.original_yaml)
  result.cleanup_keys
  render :xml => results.to_xml{:except => [:original_yaml, :updated_at]}
end
end
end

# Delete the result with the given id only if it belongs to the logged-in user.
def delete_result
  @user.send(@app_and_ver.pluralize).destroy params[:id] if params[:id]
  redirect_to :action=>:user
end
end

# Show the result in a table form
def detail
  if @app
    @alternate_names = YAML::load(@app.alternate_names)
    @flat_yaml = YAML::load(@app.flat_yaml)
    detail_method
  else
    redirect_to :controller=>@app_and_ver, :action=>:index
  end
end
end

# Show user-created charts
def detail_charts
  detail_method
end
end

# Show the original YAML (It's very readable and the heirarch is preserved)
def detail_yaml
  detail_method
end
end

private

def detail_method
  symbol = "current_#{app_and_ver}"
  session[symbol] = params[:id] if params[:id]
  if session[symbol]
    @result = @app_and_ver.camelize.constantize.find_by_id session[symbol]
    if @result
      render :partial => 'result/detail', :layout => 'main'
    else
      redirect_to :action=>:index
      end
    else
      redirect_to :action=>:index
    end
  end
end

# create a new detail(pie) chart
# Preconditions:
# @app_and_ver is set in the Application Controller
```
# @user is set in 'authorize' (as a before filter)
def create_detail_charts
  @detail_charts.reverse!
  if request.post?
    chart = DetailChart.new
    chart.name = @app_and_ver
    chart.title = params[:title]
    chart.columns = Array.new.to_yaml
    chart.user_id = @user.id
    chart.save
    redirect_to :action => :create_detail_charts
  else
    render :partial => 'result/create_detail_charts', :layout => 'main'
  end
end

# Remove a detail chart.
# Preconditions:
# @user is set in 'authorize' (as a before filter)
def remove_detail_chart
  chart = DetailChart.find params[:id]
  DetailChart.delete(params[:id]) if chart.user_id == @user.id
  redirect_to :action => :create_detail_charts
end

# Edit a detail pie chart
# Preconditions:
# @app_and_ver is set in the Application Controller
# @user is set in 'authorize' (as a before filter)
def edit_detail_chart
  # It seems like there might be a better way of keeping track of the current chart, though I think the session will always have to be used.
  symbol = "current_#{app_and_ver}_detail_chart_id".intern
  session[symbol] = params[:id] if params[:id]
  id = session[symbol].to_i

  chart = DetailChart.find(id) if id
  if chart && chart.user_id == @user.id
    @title = chart.title # for display purposes
  end

  # the columns array is stored in the database as YAML. The YAML must be converted back to a ruby array before it can be used
  @columns = YAML::load(chart.columns)

  # for each column selected by the user, we want to add that column to the columns array. If the array already had the column you tried to add, #delete it and add it again. This will merely change the order of columns.
  if request.post?
    @alternate_names.each_key do |key|
      if params[key]
        @columns.delete key if @columns.include? key
        @columns << key
      end
    end
  end

  chart.columns = @columns.to_yaml
  chart.save
end
# can't make a pie chart with only one item...
if @columns & & @columns.size > 1
  gc = GoogleChart.new
  gc.type = :pie
  gc.height = 300
  gc.width = 400
  gc.title = chart.title
end

# the way data is handled may seem obscure. We end up with something that
# looks like [[2, 4, 3]]. Pointless nesting of an array within an array?
# No. The google charts on rails code in vendor/plugins is able to
# deal with multiple datasets, and because of this it wants an array of
# for data (one array per data set). A pie chart just happens to only have
one data set.
data = [[]]
@columns.each { |c| data[0] << @flat_yaml[c] }
labels = []
@columns.each { |c| labels << @alternate_names[c] }
gc.data = data
@gc.labels = labels
@chart_url = gc.to_url # the url is used in the view
chart.url = @chart_url
chart.save
render :partial => 'result/edit_detail_chart', :layout => 'main'
else
  redirect_to :action => create_detail_charts
end

# Remove a column(pie slice) from the detail chart
# Preconditions:
# @app_and_ver is set in the Application Controller
def detail_remove_column
  # It seems like there might be a better way of keeping track of the current
  # chart, though I think the session will always have to be used.
symbol = "current_#{app_and_ver}_detail_chart_id".intern
  unless session[symbol] & & params[:id]
    redirect_to :action => create_detail_charts
  else
    column = params[:id]
id = session[symbol].to_i
chart = DetailChart.find(id) if id
if chart
columns = YAML.load(chart.columns)
columns.delete column
chart.columns = columns.to_yaml
chart.save
  end
  redirect_to :action => edit_detail_chart
end

#create a worksheet(radar/kiviat) chart
```ruby
def create_worksheet_charts
    @worksheet_charts.reverse!
    if request.post?
        chart = WorksheetChart.new
        chart.name = @app_and_ver
        chart.title = params[:title]
        chart.columns = Array.new.to_yaml
        chart.user_id = @user.id
        chart.save
        redirect_to :action => :create_worksheet_charts
    else
        render :partial => 'result/create_worksheet_charts', :layout => 'main'
    end
end

# Remove a worksheet chart
# Preconditions:
# - @user is set in 'authorize' (as a before filter)
def remove_worksheet_chart
    chart = WorksheetChart.find params[:id]
    WorksheetChart.delete(params[:id]) if params[:id] == @user.id
    redirect_to :action => :create_worksheet_charts
end

# Edit a worksheet (radar/kiviat) chart
def edit_worksheet_chart
    # It seems like there might be a better way of keeping track of the current
    # chart, though I think the session will always have to be used.
    symbol = "current_#{app_and_ver}_worksheet_chart_id".intern
    session[symbol] = params[:id] if params[:id]
    id = session[symbol].to_i
    chart = WorksheetChart.find(id) if id
    if chart
        @title = chart.title
        @columns = YAML::load(chart.columns)
        if request.post?
            @alternate_names.each_key do |key|
                if params[key]
                    @columns.delete key if @columns.include? key
                    @columns << key
            end
        end
        chart.columns = @columns.to_yaml
        chart.save
    end
    if @columns && @columns.size > 2
        gc = GoogleChart.new
        gc.type = :radar
        gc.height = 300
        gc.width = 400
        gc.title = chart.title
        starter_data = []
        @columns.each{|c| starter_data << @flat_yaml[c]}
        data = []
        #generate ficticious sample data...
        data << starter_data.map{|v| (rand*0.5)+v}
```
data << starter_data.map { |v| (rand*0.5)*v }
data << starter_data.map { |v| (rand*0.5)*v }
data << starter_data.map { |v| (rand*0.5)*v }
data[0].size.times do |i|
  # adjust the data so that the max value on each spoke is at the end of
  # the spoke and the min value is near the center of the spokes.
  temp_data = [data.collect { |d| d[i] }]
  max = temp_data.max.to_f
  min = temp_data.min.to_f
  temp_data = temp_data.map { |v| (v-min)/(max-min))*100+20 } if max != min
  data[new_val][i] = temp_data[new_val]
end

# Choose colors for the chart. Hopefully there won't be more than 9 results.
# but colors could easily be added or changed by editing this array.
available_colors = ['FF0000', '00FF00', '0000FF', 'CCCC00', '00CCCC', 'CC00CC', '999999', '444444', 'CCCCCC']
colors = []
data.size.times { |i| colors[i] = available_colors[i] }

gc.colors = colors
# add an extra piece of data to each data set to deal with a quirk of
# google charts.
data.size.times { |i| data[i] << data[i][0] }
gc.data = data
labels = []
@columns.each { |c| labels << @alternate_names[c] }
gc.labels = labels
gc.legend = [1, 2, 3, 4]
@chart_url = gc.to_url
chart.url = @chart_url
chart.save
end
render :partial => 'result/edit_worksheet_chart', :layout => 'main'
else
  redirect_to :action=>:create_worksheet_charts
end
end

def worksheet_remove_column
# It seems like there might be a better way of keeping track of the current
# chart, though I think the session will always have to be used.
symbol = "current_#{app_and_ver}_worksheet_chart_id".intern
unless session[symbol] & & params[:id]
  redirect_to :action=>:create_worksheet_charts
else
  column = params[:id]
  id = session[symbol].to_i
  chart = WorksheetChart.find(id) if id
  @columns = YAML::load(chart.columns)
  @columns.delete column
  chart.columns = @columns.to_yaml
  chart.save
  redirect_to :action=>:edit_worksheet_chart
Listing 27: Result Upload Controller

```ruby
require 'yaml'

# Allows a user to submit new results for an existing miniapplication
class ResultUploadController < ApplicationController
  layout "main"
  before_filter :authorize_user

  # Do nothing other than provide and upload field and a submit button.
  def index
    # Purpose: to read the uploaded file, see if it is of the proper format, and
    # utilize the session to store the needed data.
    def submit
      # if this method was called as a result of a file upload...
      if request.post?
        submission = params[:file]
        submission = submission.read if submission.respond_to?(:read)

        # if the file was readable...
        if submission
          # if the file contained proper YAML...
          yaml = YAML::load(submission)

          # if the YAML contained a 'Mini-Application Name' and a
          # 'Mini-Application Version'...
          app = yaml.delete('Mini-Application Name')
          version = yaml.delete('Mini-Application Version').to_s
          if app
            if version
              load "#{RAILS_ROOT}/lib/result_helper.rb"
              name = ResultHelper.generate_name app, version

              # unless this manto result already exists...
              if MiniApp.find_by_name name
                # Create flat YAML and alternate names
                load "#{RAILS_ROOT}/lib/hash_methods.rb"
                flat_yaml = yaml.flatten
                flat_yaml = flat_yaml.cleanup_keys

                # See the AdminUploadController for an explanation of the trick
                # using constants and the .send method.
                result = name.camelize.constantize.new
                flat_yaml.each_key do |column_name|
                  # This amounts to the same as, for example,
                  # SJU.students = flat_yaml['students'], but is dynamic
                  result.send( column_name:"="*flat_yaml[column_name] * )
              end
          end
      end
  end
end
```
```ruby
# Make sure to record the original YAML
result.original_yaml = submission
result.user_id = session[:user_id]

# Save the result
if result.save
  flash[:notice] = 'upload success'
  redirect_to :controller => name, :action => :detail_yaml, :id => result.id
else
  redirect_with_msg 'unable to save the result'
end
else
  redirect_with_msg "No table exists for #{app} #{version}"
end
else
  redirect_with_msg "Mini-Application Version not given"
end
else
  redirect_with_msg "Mini-Application Name not given"
end
else
  redirect_with_msg "Improper YAML format"
end
else
  redirect_with_msg "Unable to read file"
end
else
  redirect_with_msg ""
end

private

def redirect_with_msg msg
  flash[:notice] = msg
  redirect_to :action => :index
end

# Overridden from ApplicationController
def set_page_title
  @page_title = 'Result Upload' if self.action_name == "index"
end
```

Listing 28: User Controller

class Usercontroller < ApplicationController
  layout 'main'
  before_filter :authorize_user

  #since the fundamental advantage of being a user is using worksheets,
  #the index methods makes it possible for users to view their worksheets
  def index
    @worksheets = @user.worksheets
  end
end
```
It is possible for a user to edit his/her profile. Supplying the correct
old password allows the user to change email and/or password

def edit
  #if the user is trying to submit changes...
  if request.post?
    user = User.authenticate(@user.email, params[:old_password])
    #if the user got his/her old password correct...
    if user
      @user.email = params[:new_email]
      unless params[:new_password].blank?
        @user.password = params[:new_password]
        @user.password_confirmation = params[:password_confirmation]
      end
    end
    #if the user made no mistakes entering a new password or email...
    if @user.save
      flash[:notice] = "user #{@user.email} was successfully updated."
      redirect_to: action=>:index
    else
      output = "
      @user.errors.each_full{|msg| output<<"# {msg}<br/>"}
      flash.now[:notice] = output
      @user = User.find_by_id(session[:user_id])
    end
    else
      flash.now[:notice] = "Incorrect old password"
    end
  end
end

def new_worksheet
  if request.post?
    @worksheet = Worksheet.create :name => params[:name], :user_id => @user.id
    if @worksheet
      redirect_to: controller => :user, :action => :index
    end
  else
    @worksheet = Worksheet.new
  end
end

def destroy_worksheet
  session[:worksheet_id] = nil
  begin
    @worksheet = Worksheet.find(params[:id])
    if @worksheet
      flash[:notice] = 'Worksheet destroyed'
    else
      flash[:notice] = 'No Worksheet to destroy'
    end
    rescue Exception => e
      flash[:notice] = e.message
    @worksheet = nil
  end
  redirect_to: action => :index
end
Listing 29: Worksheet Controller

```ruby
# WorksheetController handles worksheets for a User.
#
# Corresponding to each result table are two public methods, one that calls the
# private method worksheet and another that calls the private method
# worksheet_remove_result. These private methods act differently depending
# on the the variable @app_and_ver, which is set to the action name, but in
# general they do the following: Worksheet checks to see if the user is adding
# results to the worksheet and updates if this is the case. It also sets up the
# necessary components for viewing charts. Worksheet_remove_result simply allows
# a result to be removed from a worksheet.

class WorksheetController < ApplicationController
  layout 'main'
  before_filter :authorize_user
  before_filter :set_app_and_ver
  before_filter :get_worksheet

  private

  # Set the @page_title
  def set_page_title
    @page_title = 'Your Worksheets' if self.action_name == "index"
    @page_title = 'Edit Profile' if self.action_name == "edit"
  end

  # Set @app_and_ver to an action name (public method) because this will be the
  # name of a result table.
  def set_app_and_ver
    @app_and_ver = self.action_name
  end

  # Used as a before filter, this method grabs the current worksheet a user is
  # working with.
  def get_worksheet
    if @user
      # if the user is arriving here by adding results to the worksheet...
      if params[:selected_worksheet]
        session[:worksheet_id]=params[:selected_worksheet][:id]
      # if the user is arriving here by selecting a worksheet through
      # the user menu...
      elsif params[:id]
        session[:worksheet_id] = params[:id]
      end
    unless session[:worksheet_id]
      flash[:notice] = "Worksheet ID needed"
      redirect_to :back
    else
```
# finally! get the worksheet.
@worksheet = Worksheet.find session[:worksheet_id], :conditions => ["user_id = ?', @user.id] unless @worksheet
flash[:notice] = "Attempt to access a worksheet that does not exist"
redirect_to :controller => :user, :action => :worksheets
end
end
end

# Do nothing. The view will show the available mini applications to choose from.
def index
def
end

# GENERATED CODE
# begin bozo_v1point0
def bozo_v1point0
  worksheet
end
def delete_bozo_v1point0
  worksheet_remove_result
def
# end bozo_v1point0

#

def yoda_v1point0
  worksheet
end
def delete_yoda_v1point0
  worksheet_remove_result
def
# END GENERATED CODE

def

private

# Mainly for viewing the results of this worksheet, but also allows a user to
# add results to a worksheet.
def worksheet
  # If a user is trying to add results to the worksheet from the 'result/
  # standard_user'
  # view...
  if request.post?
    checked = params.select { |p, result_id| p =~ /check.*\)/ } #ugly code
  if checked
    # Fancy! Since @app_and_ver stores the name of a result table, we can
# morph it into the name of the corresponding worksheet model via .camelcase and then into a constant via .constantize. Later, we can use this constant to add results from the database.

```ruby
model = "worksheet_{#{app_and_ver}}".camelcase.constantize
```

# This symbol indicates the attribute in the model (i.e., column in the table) that we need to assign a value to when creating a new entry in the model.

```ruby
symbol = "#{app_and_ver}_id".intern
```

```ruby
checked.each do |p, result_id|
  model.create( :worksheet_id => @worksheet.id ,
                symbol => result_id )
end
```

```ruby
end
```

# Setup the instance variables needed in the view

```ruby
require 'yaml'
app = MiniApp.find_by_name('@app_and_ver')
@alternate_names = YAML::load(app.alternate_names)
@columns = YAML::load(app.columns)
# Dynamically choose the result type by using @app_and_ver
@results = @worksheet.send(@app_and_ver.pluralize)
@worksheet_charts = @user.worksheet_charts.find :all , :conditions => ["name = = ?", @app_and_ver ]
render :action=>:worksheet_detail
end
```

```ruby
def worksheet_remove_result
  name = @app_and_ver.gsub!(/delete_/ , "")
  if request.post?
    if checked = params.select { |p, result_id| p =~ /^check.*$/ }
      if checked
        model = name.camelize.constantize
        checked.each do |p, result_id|
          @worksheet.send(name.pluralize).delete(model.find_by_id(result_id))
        end
      end
    end
    redirect_to :controller => :worksheet , :action => name
end
```

## E.2 Helpers

The following files can be found in `/app/helpers/`

### Listing 30: Application Helper

```ruby
# Methods added to this helper will be available to all templates in the application.
module ApplicationHelper
  def sort_td_class_helper(param)
    result = 'class="sortup"' if params[:sort] == param
    result = 'class="sortdown"' if params[:sort] == param + ' DESC'
  end
end
```

89
return result
end
def sort_link_helper(text, param)
key = param
key += " DESC" if params[:sort] == param
options = {
  :url => {:overwrite_params => {:sort => key, :page => nil}},
  :update => 'table',
}
hl_options = {
  :title => "Sort by this field",
  :href => url_for(:action => controller.action_name, :params => params.merge({:sort => key, :page => nil})),
}
link_to_remote(text, options, hl_options)
end

def gather_mini_apps
  mini_apps = MiniApp.find :all, :group => :app
  @app_and_ver_names = Hash.new
  mini_apps.each{|app| @app_and_ver_names[app.app] = Array.new }
  @app_and_ver_names.each_key do |app|
    @app_and_ver_names[app] = MiniApp.find :all, :conditions => ["app == ?", app]
  end
end
def gather_mini_apps
  mini_apps = MiniApp.find :all, :group => :app
  @app_and_ver_names = Hash.new
  mini_apps.each{|app| @app_and_ver_names[app.app] = Array.new }
  @app_and_ver_names.each_key do |app|
    @app_and_ver_names[app] = MiniApp.find :all, :conditions => ["app == ?", app]
  end
end
end

Listing 31: Result Helper

require 'yaml'
module ResultHelper
def google_pie_chart chart, alternate_names, result
  columns = YAML::load(chart.columns)
  if columns & columns.size > 1
    gc = GoogleChart.new
    gc.type = :pie
    gc.height = 300
    gc.width = 400
    gc.title = chart.title
    data = [[[]]]
Listing 32: Worksheet Helper

```ruby
require 'yaml'
module WorksheetHelper
def google_radar_chart chart, alternate_names, results
  #columns is an array which is stored in the database in YAML format
  @columns = YAML::load(chart.columns)
  #columns correspond to the 'spokes' of the radar chart, and a radar chart
  #needs at least 3 spokes.
  #Also, it makes no sense to have a radar chart with fewer than three results
  if @columns & @columns.size > 2 & results.size > 2
    gc = GoogleChart.new
    gc.type = :radar
    gc.height = 300
    gc.width = 400
    gc.title = chart.title
    #The legend shows the results
    #The labels show the names of a column value, such as "total_time"
    legend = []
    labels = []
    results.each { |result| legend << result.id }
    #the alternate_names hash keeps track of
    @columns.each { |c| labels << alternate_names[c] }
    gc.legend = legend
    gc.labels = labels
    #Populate the data array which is an array of arrays.
    #Each result is an array of values, and data is an array of these
    #result arrays. E.g. data = [[1, 2, 3],[2,1,3],[3,2,1]] where [1, 2, 3]
    #is the data for a result
    data = []
    results.size.times do |n|
      data << []
      @columns.each { |c| data[n] << results[n].send(c) }
    end
    #this code adjusts the data so that along each spoke one value (the max) will
    #lie at
    #the tip of the spoke and another value (the min) will lie near the center
    #of the spokes
    data[0].size.times do|i|
      temp_data becomes an array of a certain column value (specified by i)
      #taken from each result
end
```

91
\begin{verbatim}
43  temp_data = [data.collect{|d| d[i]}]
44  max = temp_data.max.to_f
45  min = temp_data.min.to_f
46  temp_data = temp_data.map{|v| ((v-min)/(max-min))*100+30 } if min != max
47  #reinsert the adjusted values
48  temp_data.size.times{|new_val| data[new_val][i] = temp_data[new_val] }
49
50  #A Google radar chart requires duplicating the first value for each result.
51  #For example, [2, 3, 5] becomes [2, 3, 5, 2]
52  #Comment out this line and you will see how worksheet charts change.
53  data.size.times{|n| data[n] += data[n][9] }
54  gc.data = data
55
56  #For a lack of a better way to add colors to the chart...
57  #If there are n results, then the first n colors are used.
58  available_colors = ['FF0000', '00FF00', '0000FF', 'CCCC00', '00CCCC', 'CC00CC', '999999', '444444', 'CCCCCC']
59  colors = []
60  results.size.times{|i| colors[i] = available_colors[i]}
61  gc.colors = colors
62
63  #generate the image tag with the chart url inside
64  "<img src="/"#{gc.to_url}"""
65  else
66    nil
67  end
68 end
69 end
70
\end{verbatim}

E.3 Models

The following files can be found in /app/models/

Listing 33: Admin

\begin{verbatim}
require 'digest/sha1'

class Admin < ActiveRecord::Base

  has_one :upload

  validates_presence_of :name
  validates_uniqueness_of :name

  attr_accessor :password_confirmation
  validates_confirmation_of :password
  validate :password_non_blank

  def self.authenticate(name, password)
    admin = self.find_by_name(name)
    if admin
      expected_password = Digest::SHA1.hexdigest(password)
      if admin.hashed_password != expected_password
        admin = nil
      end
    end
end
\end{verbatim}
Listing 34: Detail Chart

class DetailChart < ActiveRecord::Base
  belongs_to :user
end

Listing 35: MiniApp

# The entire app revolves around MiniApp. There should be exactly one entry in
# MiniApp for every result table. The name column is unique and refers to a
# specific version of a miniapplication. It is a combination of the name and
# version of the app; for example, SJU 1.0 has a name something like sju_v1point0.
# The original name and version are stored as well in the columns mini_name and
# mini_version. These columns are used to set up the main menu (see
# ApplicationController method gather_mini_apps and the partial
# /app/views/layouts/_main_menu.html.erb).
# MiniApp has 3 other important columns: flat_yaml, alternate_names, and columns.
# flat_yaml is exactly what it says; it is YAML with the hierarchy removed (see
# AdminUploadController method flatten_yaml). alternate_names aims to shorten
# the way column names are displayed in HTML tables (see AdminUploadController
# method edit_column_names). columns decides what columns of a result table will
# be displayed in a corresponding HTML table.
# These 3 columns are used in various controllers and views. In particular, they
# play a large role in removing code duplication. For example, all result
# controllers
# inherit from ResultController and render the same views. These views know how
# to create HTML tables specific to a result table because columns describes what
# columns to use and alternate_names tells what table headers to use (see
# /app/views/result/_header.html.erb and _info.html.erb).

class MiniApp < ActiveRecord::Base
  validates_uniqueness_of :name
end

Listing 36: Upload

class Upload < ActiveRecord::Base
  belongs_to :admin
end

Listing 37: User

require 'digest/sha1'

# This class is highly inspired by that in Agile Web Development with Rails 3rd
# edition. Slight modifications and some additions have been made.
# The password isn’t directly stored in the database. Instead, a hashed password
# is stored which can be generated time and time again if the correct password is
# supplied.
class User < ActiveRecord::Base
  # GENERATED CODE
  #begin bozo_vlogpoint0
  has_many :bozo_vlogpoint0s
  #end bozo_vlogpoint0
  #END GENERATED CODE

  has_many :worksheets
  has_many :detail_charts
  has_many :worksheet_charts

  validates_presence_of :email
  validates_uniqueness_of :email
  validates_format_of :email,
    :with => /^[\A[\-\s]+\0((?:[\-a-z0-9]+)+[\-a-z]{2,})\Z/i
    , :message => "email format must be valid"
attr_accessor :password_confirmation
  validates_confirmation_of :password
  validate :password_non_blank

  def self.authenticate(name, password)
    user = self.find_by_name(name)
    if user
      expected_password = encrypted_password(password, user.salt)
      if userhashed_password != expected_password
        user = nil
    end
    end
    user
  end

94
Listing 38: Worksheet

class Worksheet < ActiveRecord::Base
  belongs_to :user

  # GENERATED CODE
  # begin bozo_v1point0
  has_many :worksheet_bozo_v1point0s
  has_many :bozo_v1point0s, :through => :worksheet_bozo_v1point0s
  # end bozo_v1point0

  # END GENERATED CODE

end

Listing 39: Worksheet Chart

class WorksheetChart < ActiveRecord::Base
  belongs_to :user
end

E.4 Views

E.4.1 Admin Login Views

The following files can be found in /app/views/admin_login/
E.4.2 Admins Views

The following files can be found in /app/views/admins/

Listing 40: Admin Login Views - Index

```html
<div id="login">
  <form_tag do %>
    <p>
      <label for="name" class='mantovo-label'>Name:</label>
      <text_field_tag :name, params[:name] %>
    </p>
    <p>
      <label for="password" class='mantovo-label'>Password:</label>
      <password_field_tag :password, params[:password] %>
    </p>
    <p>
      <submit_tag "Login" %>
    </p>
  </form_tag>
</div>
```

Listing 41: Admins Views - Edit

```html
<h1>Editing admin</h1>
<form_for @admin do |f| %>
  <% f.error_messages %>
  <p>
    <% f.label :name %> <br />
    <% f.text_field :name %>
  </p>
  <% f.label :password %> <br />
  <% f.text_field :password %>
  <% f.label :password_confirmation %> <br />
  <% f.text_field :password_confirmation %>
  <% f.submit "Update" %>
</form_for>
<p>
  <%= link_to 'Back', admins_path %>
</p>
```

Listing 42: Admins Views - Index

```html
<h1>Listing admins</h1>
<table>
  <tr>
    <th>Name</th>
  </tr>
```
Listing 43: Admins Views - New

```
<% form_for(@admin) do |f| %>
  <p>
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </p>
  <p>
    <%= f.label :password %><br />
    <%= f.text_field :password %>
  </p>
  <p>
    <%= f.label :password_confirmation %><br />
    <%= f.text_field :password_confirmation %>
  </p>
  <%= f.submit "Create" %>
<% end %>
<% link_to 'Back', admins_path %>
```

E.4.3 Admin Upload Views

The following files can be found in /app/views/admin_upload/

Listing 44: Admin Upload Views - Column Selection

```
<p class='instructions'>
  This is a sample table to show how results of this new mini-app and version will be displayed.
</p>
<table class='result-table'>
<tr>
```
97
Listing 45: Admin Upload Views - Edit Column Names

This option is given because names like "time_sparsemv_total_time_overhead" are hard to read and crowd table columns (move on to <%= link_to "column selection", :action => :column_selection %>
if you want proof of this).

Note: these names will only affect display, not the database.

<form_tag do %>
<table>
  <% @alternate_names.keys.sort_each do |key| %>
    <tr>
      <% unless key %>
        <td>
          <label>
            <%= @alternate_names[key] %>
          </label>
        </td>
      <% end %>
    </tr>
  <% end %>
</table>
</form_tag>

<br/>
<%= link_to "Next: Review", :action => :review %>
<br/>
<%= link_to "Back: Edit Column Names", :action => :edit_column_names %>

<h3> Edit Column Names </h3>

<p class='instructions'>
  This option is given because names like "time_sparsemv_total_time_overhead" are hard to read and crowd table columns (move on to <%= link_to "column selection", :action => :column_selection %>
if you want proof of this).

Note: these names will only affect display, not the database.
</p>

<form_tag do %>
<table>
  <% @alternate_names.keys.sort_each do |key| %>
    <tr>
      <% unless key %>
        <td>
          <label>
            <%= @alternate_names[key] %>
          </label>
        </td>
      <% end %>
    </tr>
  <% end %>
</table>
</form_tag>
### Listing 46: Admin Upload Views - Finish Upload

```erb
<%=
  h1
  "#{app} #{version}"
%>

<%= h2 %>
  Upload Finished
<%= /h2 %>

<%= p %>
  File named "#{filename}" was added to /result
<%= /p %>

<%= h3 %>
  How to activate the changes
<%= /h3 %>

<%= pre %>
  rake result:add[#{filename}]
<%= /pre %>

To remove the app, run
<%= pre %>
  rake result:remove[#{filename}]
<%= /pre %>

Note that some shells require escaping the brackets with \\ and \\.

<%= br %>

<%= h3 %>
  Then, <%= link_to "follow this link", controller => @filename, action => :index %>
<%= /h3 %>

<% end %>
```

### Listing 47: Admin Upload Views - Index

```erb
<p class='instructions'>
  Please select the YAML file you wish to upload (uploading the file will merely advance you to the next step of the process for creating a new mini-app or an old mini-app with a new version.)
<%= br %>

You are beginning the process of creating a new table in the database and building the framework for inserting into and viewing data from that table. You will be guided through the process every step of the way.
<%= /p %>

<%= form_tag( { action => :submit }, { multipart => true } ) do %>
  <%= file_field_tag "file" %>
<%= submit_tag "SUBMIT" %>
<% end %>
```
The information you have just provided can be changed at a later date because it only affects the way data is viewed, **not** how it’s stored.

What you can’t change is the format of the YAML. The database table itself is rigid. If you want to make changes to the data a mini-application gathers, you will need to give the app a new version and go through this process again thereby creating another table in the database.

```erb
<p class='instructions'>

The names you have provided will affect the following:

- In a table view of results, these names will serve as the headers as shown below.
- In any detail (pie) chart that a user creates, these names will label the pie slices.
- In any worksheet (radar/kivi) chart that a user creates, these names will label the spokes.

```yaml
<table>
<thead>
<tr>
<th>key</th>
<th>alternate_names[key]</th>
</tr>
</thead>
</table>
```

The table below shows you which pieces of data you would like to display in a table view of results.

```erb
<table class='result-table'>

<table>
<thead>
<tr>
<th>ID</th>
<th>details</th>
</tr>
</thead>
</table>
```

```ruby
<br/>
```
<br/>
```

<h3>Alternative Names</h3>

```erb
<br/>
```
<br/>
```

<h3>Sample Table</h3>

```erb
<br/>
```
<br/>
E.4.4 Layouts Views

The following files can be found in /app/views/layouts/

Listing 49: Layouts Views - Admins

```
<DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="content-type" content="text/html;charset=UTF-8" />
    <title>Admin: <% controller.action_name %></title>
    <% stylesheet_link_tag 'main' %>
</head>
<body>
<div id='admin-nav'>
    <span><%= link_to "Home", :controller => :result %></span>
    | 
    <span><%= link_to "Admin Home", :controller => :admins %></span>
    | 
    <span><%= link_to "Upload", :controller => :admin_upload %></span>
    | 
    <span><%= link_to "Logout", :controller => :admin_login, :action => :logout %></span>
</div>
<div id='upload-nav'>
    <span><%= link_to "Edit Column Names", :controller => :admin_upload, :action => :edit_column_names %></span>
    | 
    <span><%= link_to "Column Selection", :controller => :admin_upload, :action => :column_selection %></span>
    | 
    <span><%= link_to "Review", :controller => :admin_upload, :action => :review %></span>
</div>
<br/>
<p style="color: green"><%= flash[:notice] %></p>
</body>
</html>
```

Listing 50: Layouts Views - Header partial

```
<div id='header'>
    <span id='header-left'>
        <% link_to "home", {:controller => :result, :action => :index} %>
        <% if session[:user_id] %>
```

101
Listing 51: Layouts Views - Main

```ruby
<%= link_to "worksheets", :controller=>:user%>
<%= link_to "upload", :controller=>:result_upload%>
<% end %>
</span>

<% if user = User.find_by_id(session[:user_id]) %>
<%= link_to "Edit Profile", :controller=>:user, :action=>:edit %>
<%= link_to "logout #{user.email}", {controller=>:login, :action=>:logout}%
<% else %>
<%= link_to "login", {controller=>:login, :action=>:index}%
<% end %>
</span>
</div>
```

Listing 52: Layouts Views - Main Menu partial

```ruby
<% gather_mini_apps %>
<br/>
<ul id="nav">
  <% @app_and_ver_names.each_key do |name| %>
  <li><a href="#" name="%"></a>
  <ul>
    <% @app_and_ver_names[name].each do |app| %>
    <li><a href="\/#{app.name}/" version="%"></a></li>
    <% end %>
  </ul>
  <% end %>
</ul>
<br id="nav-floor" />
```
Listing 53: Layouts Views - Menu Javascript partial

```javascript
<script type="text/javascript">
    sfHover = function() {
        var sfEl = document.getElementById("nav").getElementsByTagName("LI");
        for (var i = 0; i < sfEl.length; i++) {
            sfEl[i].onmouseover = function() {
                this.className += " sfhover";
            }
            sfEl[i].onmouseout = function() {
                this.className = this.className.replace(new RegExp(" sfhover\\b"), "");
            }
        }
        if (window.attachEvent) window.attachEvent("onload", sfHover);
    }
</script>
```

E.4.5 Login Views

The following files can be found in /app/views/login/

Listing 54: Login Views - Index

```html
<div id="login">
 <p><h3>Login</h3></p>
 <p><label for="email" class='mantevo-label'>Email:</label>
 <input type="text" name="email" value="" size=>25, class => 'mantevo-input' />
</p>
 <p><label for="password" class='mantevo-label'>Password:</label>
 <input type="password" name="password" class => 'mantevo-input' />
</p>
 <p><input type="submit" value="Login", class => 'mantevo-submit' />
</p>
</div>
```

Listing 55: Login Views - Register

```html
<div class='mantevo-form'>
 <p class='instructions'>Registering will allow you to upload results and use worksheets to analyse results.</p>
 <p><label for="user_email" class='mantevo-label'>Email:</label>
 <input type="text" name="email", size => 25, class => 'mantevo-input' />
</p>
```

103
E.4.6 Result Views

The following files can be found in /app/views/result/

Listing 56: Result Views - Create Detail Charts partial

```hll
<br/>
<h3>Create new chart</h3>
<form_tag :action=>'create_detail_charts' do -%>
  <label for='title'>Chart Title</label>
  <% text_field_tag 'title' %>
  <% submit_tag 'Create' %>
</form_tag>
<tr>
  <td><%= chart.title %></td>
  <td><%= img src='#{chart.url}' %></td>
  <td><%= link_to 'edit', {id => chart.id, :action => 'edit_detail_chart'} %></td>
  <td><%= link_to 'delete', {id => chart.id, :action => 'remove_detail_chart'} %></td>
</tr>
@end
</table>
```

Listing 57: Result Views - Create Worksheet Charts partial

```hll
<br/>
<h3>Create new worksheet chart</h3>
<form_tag :action=>'create_worksheet_charts' do -%>
  <label for='title'></label>
</form_tag>
```
Listing 58: Result Views - Detail partial

```ruby
<% if controller.action_name == 'detail_charts' %>
  <% if @user %>
    <br/>
  <% end %>
  <% @detail_charts.each do |chart| %>
    <%= "<img src='<%= google_pie_chart_chart, @alternate_names, @result %>'>" >
  <% end %>
<% end %>
<% else %>
  <%= link_to 'login', {:controller=>:login}, :class=>'white-txt' %>
<% end %>
```
<% if controller.action_name == 'detail' %>
  <div class='result-table'>
    <table>
      <% @flat_yaml.each_key do |col_name| %>
      <tr class='line-odd', 'line-even'>
        <td>% @alternate_names[col_name] %</td>
        <td>% @result.send col_name %</td>
      </tr>
      </% end %>
    </table>
  </div>
</% if controller.action_name == 'detail_yaml' %>

Listing 59: Result Views - Edit Detail Chart partial

<samp>
<% link_to 'back to detail charts', :action=>:create_detail_charts %>
<br/>
<h2>% @title %</h2>
<% if @columns && @columns.empty? %>
  <% if @columns.size == 1 %>
    <p>Click links to remove items from chart.</p>
  <% end %>
</% if @columns && @columns.empty? %>
<table class='result-table'>
  <tr>
    <% @columns.each do |val| %>
    <th>% link_to "#{h @alternate_names[val]}", {:action => :detail_remove_column , :id => "#{h val}"} %</th>
  </tr>
</% if @columns && @columns.empty? %>
</table>
</% if @columns && @columns.empty? %>
</samp>
Listing 60: Result Views - Edit Worksheet Chart partial

```html
1-\n<br/>
2-\n% end%
3-\n% submit_tag 'Add'
4-\n% end%
5-\n% if @chart_url%
6-\n<h2>Chart: <%= @title %></h2>
7-\n<p>
8-\n<%= "<img src="/#{@chart_url}" /" %>
9-\n</p>
10-\n% end%
11-\n
12-\n% if @columns && !@columns.empty?%
13-\n<% if @columns.size < 3 %>
14-\nClick links to remove items from chart.
15-\n</p>
16-\n</table class='result-table'>
17-\n<tr>
18-\n<% @columns.each do |val| %>
19-\n<th><%= link_to "#{@alternate_names[val]}", {:action => :worksheet_remove_column, :id => "#{h val}"} %></th>
20-\n<% end %>
21-\n</tr>
22-\n</tr class='line-even'>
23-\n<% @columns.each do |val| %>
24-\n<td><%= @flat_yaml[val] %></td>
25-\n<% end %>
26-\n</tr>
27-\n</table>
28-\n% end%
29-\n% form_tag do
30-\n<% @flat_yaml.keys.sort.each do |key| %>
31-\n<% if @flat_yaml[key].class != String %>
32-\n<label><%= @alternate_names[key] %></label>
33-\n<% if check_box_tag "#{key}" %>
34-\n<br/>
35-\n<% end %>
36-\n<% end %>
37-\n<% end %>
38-\n<%= submit_tag 'Add' %>
39-\n```

107
Listing 61: Result Views - Header partial

```ruby
<th <%= sort_td_class_helper 'id' %>>
  <%= sort_link_helper 'ID', 'id' %>
</th>

<% if @columns.class == Array %>
  <% @columns.each do |col| %>
    <th <%= sort_td_class_helper col %>>
      <%= sort_link_helper @alternate_names[col], col %>
    </th>
  <% end %>
</% end %>
```

Listing 62: Result Views - Home

```
Homepage: What do we want the homepage to look like?
```

Listing 63: Result Views - Info partial

```ruby
<td>= h(@result.id) %></td>
  <% @columns.each do |col| %>
    <td>= h(@result.send(col)) %></td>
  <% end %>
```

Listing 64: Result Views - Results partial

```ruby
<% if @results && @results.size > 0 %>
  <% if session[:user_id] && controller.controller_name != 'worksheet' %>
    <%= link_to "view all results", :controller=>@app_and_ver, :action=>:index %>
  <% else %>
    <%= link_to "view my results only", :controller=>@app_and_ver, :action=>:user %>
  <% end %>
<% else %>
  <%= render :partial=> 'result/worksheet', :object=>@results %>
<% elsif session[:user_id] && controller.controller_name == 'worksheet' %>
  <%= render :partial=> 'result/standard_user', :object=>@results %>
<% else %>
  <%= render :partial=> 'result/standard', :object=>@results %>
<% end %>
<% elsif controller.controller_name == 'worksheet' %>
  <p>No Data</p>
<% elsif controller.controller_name == '#{controller.action_name}' %>
  <p>No Data</p>
<% else %>
  <%= link_to 'collect data', :controller=>"#{controller.action_name}" %>
  <p>No Data</p>
<% end %>
```

108
Listing 65: Result Views - Standard partial

```html
<% will_paginate @results %>

<div class='result-table'>
  <table>
    <tr>
      <%= render :partial => "result/header" %>
    </tr>

    <% for @result in @results %>
      <tr class=<% cycle 'line-odd', 'line-even' %> >
      <td>
        <%= render( :partial => "result/info", :object => @result ) %>
      </td>
      <td>
        <%= link_to 'Detail', {:id => @result.id, :controller=>'#{@app_and_ver}','action'=>:detail_yaml} %>
      </td>
      <% if controller.action_name == 'user' %>
      <td>
        <%= link_to 'Destroy', {:id => @result.id, :controller=>'#{@app_and_ver}','action'=>:delete_result }, :confirm=>'Are you sure?' %>
      </td>
      <% end %>
    </tr>
    <% end %>
  </table>
</div>
```

Listing 66: Result Views - Standard User partial

```html
<% will_paginate @results %>

<div class='result-form'>
  <%= form_tag({:controller => :worksheet, :action => @app_and_ver}) do %>

  <div class='result-table'>
    <table>
      <th></th>
      <tr>
        <%= render :partial => "result/header" %>
      </tr>

      <% for @result in @results %>
      <tr class=<% cycle 'line-odd', 'line-even' %> >
      <td>
        <%= checkbox_tag "check_#{@result.id}" , @result.id %>
      </td>
      <td>
        <%= render :partial => "result/info", :object => @result %>
      </td>
      <td>
        <%= link_to 'Detail', {:id => @result.id, :controller=>'#{@app_and_ver}','action'=>:detail_yaml} %>
      </td>
      <% if controller.action_name == 'user' %>
      <td>
        <%= link_to 'Destroy', {:id => @result.id, :controller=>'#{@app_and_ver}','action'=>:delete_result }, :confirm=>'Are you sure?' %>
      </td>
      <% end %>
    </tr>
    <% end %>
  </table>
</div>
```
You have not created a worksheet yet.

Choose worksheet

Choose worksheet

Add To Worksheet

Remove From Worksheet

Listing 67: Result Views - Worksheet partial
E.4.7 Result Upload Views

The following files can be found in /app/views/result_upload/

Listing 68: Result Upload Views - Index

```erb
<% form_tag({:action => :submit}, {:multipart => true}) do -%>
  <label for="file">results file</label>
  <%= file_field_tag "file" %>
  <%= submit_tag "SUBMIT" %>
<% end %>
</div>
```

E.4.8 User Views

The following files can be found in /app/views/user/

Listing 69: User Views - Edit

```erb
<% form_tag do -%>
  <p>
    <label for="old_password">Old Password:</label>
    <%= password_field_tag :old_password, nil, :size => 25 %> 
  </p>
  <p>
    <label for="new_email">New Email:</label>
    <%= text_field_tag :new_email, @user.email, :size => 25 %>
  </p>
  <p>
    <label for="new_password">New Password:</label>
    <%= password_field_tag :new_password, nil, :size => 25 %> 
  </p>
  <p>
    <label for="password_confirmation">Confirm New Password:</label>
    <%= password_field_tag :password_confirmation, nil, :size => 25 %>
  </p>
  <%= submit_tag "Update", :confirm => "Are you sure?" %>
<% end %>
</div>
```

Listing 70: User Views - Index

```erb
<% if @worksheets.size > 0 %>
  <%= render(:partial => "user/worksheets", :object => @user) %>
<% else %>
  <%= link_to "new worksheet", :controller => :user, :action => :new_worksheet %>
<% end %>
```

111
Please `%=` link_to "create a new worksheet", :controller => :user, :action => :new_worksheet `%`.

Listing 71: User Views - New Worksheet

```
<% form_tag do %>
  <p>
    <label for="name">Name:</label>
    <%= text_field_tag :name, params[:name], :size => 25 %>
  </p>
  <%= submit_tag "Create" %>
</% end %>
```

Listing 72: User Views - Users partial

```
<% for user in @users %>
  <tr class="<%= cycle ("admin-line-odd_user", "admin-line-even_user") %>">
    <td><%= link_to h(user.email), {controller => :user, :action => :user_detail}, :id => user.id %></td>
    <td><%= user.submissions.count %></td>
    <td><%= link_to 'Destroy', {id => user.id, :action => 'destroy_user'}, :confirm => 'Are you sure?', :method => :delete %></td>
  </tr>
<% end %>
```

Listing 73: User Views - Worksheets partial

```
<h2>Your Worksheets</h2>
<% div class='result-table' %>
<table>
  <tr>
    <th>Name</th>
  </tr>
```

112
E.4.9 Worksheet Views

The following files can be found in /app/views/worksheet/

Listing 74: Worksheet Views - Index

```erb
<% if @worksheets %>
<tr class="<% = cycle('line-odd', 'line-even') %>">
  <td><%= link_to h(@worksheet.name), {:controller => :worksheet, :action => :index, :id => @worksheet.id} %></td>
  <td><%= link_to 'Destroy', {:id => @worksheet.id, :action => 'destroy_worksheet'}, :confirm => 'Are you sure?', :method => :delete %>
</tr>
<% end %>
</table>
</div>
```

Listing 75: Worksheet Views - Worksheet partial

```erb
<% @app_and_ver_names.each_key do |name| %>
  <% @app_and_ver_names[name].each do |app| %>
    <% = render(:partial => "worksheet/worksheet", :object => @worksheet) %>
  <% end %>
<% end %>
```

Listing 76: Worksheet Views - Worksheet Detail

```erb
<% app = MiniApp.find_by_name @app_and_ver %>
<% = render(:partial => 'result/results', :object => @results) %>
<% = link_to 'create/edit charts', {:controller=>@app_and_ver, :action=>:create_worksheet_charts}, :target=>'_blank' %>
<% if @results.size > 2 %>
```
E.5 Rake Task

The following files can be found in /lib/tasks/

Listing 77: Result.Rake

```ruby
# If either of these tasks should fail and it becomes necessary to clean up the
# app by hand here are some instructions.
#
# INSTRUCTIONS FOR MANUALLY REMOVING A RESULT (hopefully this will never happen)
#
# You will first and foremost need to know the name of the result you're trying
# to remove. It is the name of the file in /results. If this file no longer
# exists for some reason and if you still know the appname and version name,
# (check a YAML result file if you don't know them), do the following to get the
# name:
# - open up a terminal and navigate to the root directory of the app
# - run script/console to open up an irb session with the rails environment
#   already loaded. Type the following commands:
#   require "lib/result_helper.rb"
#   ResultHelper.generate_name app, version
#   The output is the name you should use.
#
# - remove inserted code from the following files
# - /app/controllers/worksheet_controller.rb
```
# Looking at these files, it should be obvious what code to delete. Look for
# a comment with "GENERATED CODE" in it, then find the code pertaining to the
# manteno result you're trying to remove and delete it. This code will be,
# enclosed by two comments of the form "#begin name" and "#end name"
#
# remove 3 generated files
# - /app/models/name.rb
# - /app/models/worksheet_name.rb
# - /app/controllers/name_controller.rb
#
# remove the manteno result from MiniApps table
# - navigate to the root directory of the app and run 'script/console'
# - now you can type the following ruby code to delete the app from the table
#   MiniApp.find_by_name(name).destroy
#
# Now things get a little trickier with migrations. See /lib/result_helper method
# 'create_delete_migrations' for more info. (knowledge of how to create migrations
# and the conventions that must be followed is assumed)
#
# create 2 new migrations of the form
# class DeleteName < ActiveRecord::Migration
#   def self.up
#     begin
#       drop_table :name
#     rescue
#     end
#   end
# end
#
# One of these migrations is for the regular manteno result table and the other
# is for the worksheet table. For the worksheet table, the migration will look
# slightly different:
# class DeleteWorksheetName < ActiveRecord::Migration
#   def self.up
#     begin
#       drop_table :worksheet_name
#     rescue
#     end
#   end
# end
#
# run the migrations with rake db:migrate
#
# remove all migrations with filenames containing the manteno result-type name
# require 'yaml'
# require 'lib/result_helper.rb'

namespace :result do
desc "Add a manteno result to the app"
task :add, :filename, :needs => [:environment] do |t, args|
  args.with_defaults[:filename] = nil
  unless args.filename
    puts "please run rake result:add[filename]"
  else

end
path = "results/#{args.filename}"
yaml = File.read "#{RAILS_ROOT}/#{path}" if File.exist? "#{RAILS_ROOT}/#{path}"
upload = Upload.new
upload = YAML::load(yaml) if yaml

if upload
  puts "generating #{upload.app} #{upload.version} code ..."
  ResultHelper.insert_code upload
  ResultHelper.create_migration_files upload
  ResultHelper.create_regular_files upload
  ResultHelper.new_mini_app_entry upload
  # Run the newly created migrations
  Rake::Task["db:migrate"].invoke
  # TODO If the environment is Development, run tests...
  # TODO write tests
end

desc "Remove a manttevo result from the app"
task :remove, :filename, :needs => [:environment] do |t, args|
  args.with_defaults(:filename => nil)
  unless args.filename
    puts "please run rake result:remove[filename]"
  else
    path = "results/#{args.filename}"
yaml = File.read "#{RAILS_ROOT}/#{path}" if File.exist? "#{RAILS_ROOT}/#{path}"
upload = Upload.new
upload = YAML::load(yaml) if yaml

    if upload
      puts "removing #{upload.app} #{upload.version} code ..."
      ResultHelper.remove_inserted_code upload
      ResultHelper.remove_regular_files upload
      ResultHelper.remove_mini_app_entry upload
      ResultHelper.create_delete_migrations upload
      # Run the newly created migrations (these migration drop the tables)
      begin
        Rake::Task["db:migrate"].invoke
        rescue Exception => e
          puts 
        puts "\nTry running rake result:remove again to complete removal\n"
        ensure
          # Destroy all migration files
          ResultHelper.cleanup_migrations upload
        end
      # TODO If the environment is Development, run tests...
      # TODO write tests
      else
        puts "unable to load #{path}"
      end
    end
  end
end
end
end
end
end

desc "Remove a manttevo result from the app"
task :remove, :filename, :needs => [:environment] do |t, args|
  args.with_defaults(:filename => nil)
  unless args.filename
    puts "please run rake result:remove[filename]"
  else
    path = "results/#{args.filename}"
yaml = File.read "#{RAILS_ROOT}/#{path}" if File.exist? "#{RAILS_ROOT}/#{path}"
upload = Upload.new
upload = YAML::load(yaml) if yaml

    if upload
      puts "removing #{upload.app} #{upload.version} code ..."
      ResultHelper.insert_code upload
      ResultHelper.inserted_code upload
      ResultHelper.create_regular_files upload
      ResultHelper.new_mini_app_entry upload
      ResultHelper.remove_inserted_code upload
      ResultHelper.remove_regular_files upload
      ResultHelper.remove_mini_app_entry upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelper.inserted_code upload
      ResultHelp
### E.6 Configuration

The following files can be found in `/config/`

**Listing 78: Database YAML**

```yaml
# SQLite version 3.x
# gem install sqlite3-ruby (not necessary on OS X Leopard)
development:  
    adapter: sqlite3  
    database: db/development.sqlite3  
    timeout: 5000

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:  
    adapter: sqlite3  
    database: db/test.sqlite3  
    timeout: 5000

production:  
    adapter: sqlite3  
    database: db/production.sqlite3  
    timeout: 5000
```

### E.7 Migrations

The following files can be found in `/db/migrate/`

**Listing 79: Create Users**

```ruby
class CreateUsers < ActiveRecord::Migration
def self.up  
    create_table :users do |t|  
        t.string :email  
        t.string :hashed_password  
        t.string :salt  
        t.timestamps  
    end  
end
def self.down  
    drop_table :users  
end
end
```

**Listing 80: Create Worksheets**

```ruby
class CreateWorksheets < ActiveRecord::Migration
def self.up  
    create_table :worksheets do |t|  
        t.integer :user_id  
        t.string :name  
    end
end
```
Listing 81: Create Admins

class CreateAdmins < ActiveRecord::Migration
  def self.up
    create_table :admins do |t|
      t.string :name
      t.string :hashed_password
      t.timestamps
    end
    Admin.create :name => 'mantevo', :password => 'trilinos'
  end
  def self.down
    drop_table :admins
  end
end

Listing 82: Create MiniApps

class CreateMiniApps < ActiveRecord::Migration
  def self.up
    create_table :mini_apps do |t|
      t.string :app
      t.string :version
      t.string :name
      t.text :flat_yaml
      t.text :alternate_names
      t.text :columns
      t.timestamps
    end
  end
  def self.down
    drop_table :mini_apps
  end
end

Listing 83: Create Sessions

class CreateSessions < ActiveRecord::Migration
  def self.up
    create_table :sessions do |t|
  end
end
Listing 84: Create Uploads

```ruby
class CreateUploads < ActiveRecord::Migration
  def self.up
    create_table :uploads do |t|
      t.string :name
      t.string :app
      t.string :version
      t.text :flat_yaml, :default => ''
      t.text :alternate_names, :default => ''
      t.text :columns, :default => ''
      t.integer :admin_id
      t.timestamps
    end
  end
  def self.down
    drop_table :uploads
  end
end
```

Listing 85: Create Detail Charts

```ruby
class CreateDetailCharts < ActiveRecord::Migration
  def self.up
    create_table :detail_charts do |t|
      # The name column will be used to distinguish between mini_apps and versions
      # For example the name might be hpcga_v1point0
      t.string :name
      t.text :columns
      t.string :title
      t.string :url
      t.integer :user_id
      t.timestamps
    end
  end
  def self.down
    drop_table :detail_charts
  end
end
```
Listing 86: Create Worksheet Charts

```
class CreateWorksheetCharts < ActiveRecord::Migration

  def self.up
    create_table :worksheet_charts do |t|
      # The name column will be used to distinguish between mini_apps and versions
      # For example the name might be kpecg_v1point0
      t.string :name
      t.text  :columns
      t.string :title
      t.string :url
      t.integer :user_id
      t.timestamps
    end
  end

  def self.down
    drop_table :worksheet_charts
  end
end
```

E.8 Other

E.8.1 Lib

The following files can be found in /lib/

Listing 87: Hash Methods

```
# Add some useful functionality to the Hash class
class Hash
  def flatten
    flattened_hash = Hash.new
    flattened_myself self, ""
    return flattened_hash
  end

  def cleanup_keys
    # TODO this should do more than remove whitespace
    temp_hash = Hash.new
    self.each_key do |key|
      if self[key].class == String || self[key].class == Float || self[key].class == Fixnum
        name = key.gsub(/\s+/, '_')
        temp_hash[name] = self[key]
      elsif self[key] == Hash
        temp_hash[name] = self[key].cleanup_keys
      end
    end
  end
end
```
# If the current hash is flat, it returns a hash of the form
# { :name1 => :name1, :name2, => :name2, ... :namen => :namen }
def generate_alternate_names
  alternate_names = Hash.new
  self.each_key do |key|
    alternate_names[key] = key
  end
  return alternate_names
end

private

# This recursive method populates the instance variable @flattened_hash
def flatten_myself elem, key
  if elem.class == Hash
    key << "" if key != ""
    elem.each_key do |child| flatten_myself elem[child], key + child
  elsif elem.class == Fixnum || elem.class == Float || elem.class == String
    @flattened_hash[key] = elem
  end
end
end

Listing 88: Result Helper

require 'find'

# Serves to aid the result.rake task in adding and removing mantevo results
module ResultHelper
  def self.generate_name app, version
    # name is used for creating a class name. To make it unique
    # join the app with the version via an underscore.
    # An error occurs when numbers immediately follow an underscore
    # in a class name, hence the v after the underscore.
    name = app + "\v" + version
    if name.index "."
      name = name.gsub(/\./, "point").downcase
    else
      name << "point0"
    end
    return name
  end

  # Insert the code for the given upload
  def self.insert_code_upload
    # Hash containing paths (files) as keys pointing to output-worthy names
    files = {
      "/app/controllers/worksheet_controller.rb" => 'worksheet controller',
      "/app/models/worksheet.rb" => 'worksheet model',
      "/app/models/user.rb" => 'user model'}
# Hash containing paths (files) as keys pointing the text to insert
codes = {
    "/app/controllers/worksheet_controller.rb" => self.
        edit_worksheet_controller(upload.name),
    "/app/models/worksheet.rb" => self.edit_worksheet_model(upload.name),
    "/app/models/user.rb" => self.edit_user_model(upload.name)
}

# To identify the correct place to insert new code
finder = 'GENERATED CODE'

# Insert generated code into each file
files.each_key do |file|
    text_to_insert = codes[file]
    body = File.read("#{RAILS_ROOT}/#{file}")
    unless body.index "begin #{upload.name}"
        i = body.index finder
        body.insert([(i+finder.length), text_to_insert])
        File.open("#{RAILS_ROOT}/#{file}", 'w') { |f| f.write body }
    else
        puts "exists #{upload.app} #{upload.version} #{files[file]} code"
    end
end
end

# Because migration files have a timestamp, creating and removing these files
# is a bit tricky

def self.create_migration_files upload
    # Migration filenames need a time associated with them
    time = Time.now.utc.strftime "%Y%m%d%H%M"
    # The end of the file names without a timestamp... use these to compare to
    # existing files.
    migration_name = "_create_#{upload.name.pluralize}.rb"
    worksheet_migration_name = "_create_worksheet_#{upload.name.pluralize}.rb"
    # Prepare the two migrations in a hash
    files = {
        "/db/migrate/#{time}/#{migration_name}" => self.new_migration(upload),
        "/db/migrate/#{(time.to_i + 1).to_s}/#{worksheet_migration_name}" =>
            self.new_worksheet_migration(upload.name)
    }
    # Iterate through the hash
    files.each_key do |file|
        # Does a similar file already exist? Initially, the answer is no...
        exists = nil
        # Because times on the names are unreliable, just look at the end of the
        # file string. Check all migrations and see if any match the one
        # we're trying to create...
        Find.find("#{RAILS_ROOT}/db/migrate") do |migration|
            if FileTest.directory?(migration)
                Find.prune if migration.match /\.*\.svn\.*/
            else
                # prepare the strings for comparison
                end_of_migration = migration.gsub RAILS_ROOT, ""
                index = end_of_migration.index("_")
    end
# if a file in the migration directory is of the wrong form, get
# on with life.

next unless index

end_of_migration = end_of_migration[index, (end_of_migration.length -
index)]
index = file.index(" ")
end_of_filename = file[index, (file.length - index)]

# if match is found, move on to the next file ...
if end_of_migration == end_of_filename
  exists = true
  puts "exists #{file}"
  break;
end
end

unless exists
  File.open("#{RAILS_ROOT}/#{file}", 'w') do |f|
    f.write files[file]
  end
  puts "created #{file}"
end
end
end

# Create files for the given upload

def self.create_regular_files upload
  # Create Controllers and Models
  files = {
    "#{RAILS_ROOT}/#{file}" => self.new_model(upload.name),
    "#{RAILS_ROOT}/#{file}" => self.new_worksheet_model(upload.name),
    "#{RAILS_ROOT}/#{file}" => self.new_controller(upload.name)
  }
  files.each_key do |file|
    if File.exist? "#{RAILS_ROOT}/#{file}"
      puts "exists #{file}"
    else
      File.open("#{RAILS_ROOT}/#{file}", 'w') do |f|
        f.write files[file]
      end
      puts "created #{file}"
    end
  end
end
end

def self.new_mini_app_entry upload
  # Last, but certainly not least, insert into the MiniApp table
  unless MiniApp.find_by_name upload.name
    miniapp = MiniApp.new
    miniapp.name = upload.name
    miniapp.app = upload.app
    miniapp.version = upload.version
    miniapp.flat_yaml = upload.flat_yaml
    miniapp.alternate_names = upload.alternate_names
    miniapp.columns = upload.columns
    miniapp.save
  end
end
puts "inserted #{upload.app} #{upload.version} in the MiniApp table"
else
  puts "exists #{upload.app} #{upload.version} in the MiniApp table"
end

```ruby
def self.remove_inserted_code_upload
  files = {"/app/controllers/worksheet_controller.rb" => 'worksheet controller',
           "/app/models/worksheet.rb" => 'worksheet model',
           "/app/models/user.rb" => 'user model'}
  # Remove the portions of the files between the starting index and
  # the ending index
  files.each_key do |file|
    body = File.read "#{RAILS_ROOT}/#{file}"
    starting_index = body.index "begin #{upload.name}"
    ending_index = body.index "end #{upload.name}" + "end #{upload.name}".length
    if starting_index & ending_index
      beginning = body[0,starting_index - 1]
      ending = body[ending_index, [body.length - ending_index]]
      body = beginning + ending
      File.open("#{RAILS_ROOT}/#{file}", 'w') { |f| f.write body }
      puts "removed #{files[file]} code for #{upload.app} #{upload.version}"
    else
      puts "non-existent #{files[file]} code for #{upload.app} #{upload.version}"
    end
  end
end

```ruby
def self.remove_regular_files_upload
  files = ["/app/models/#{upload.name}.rb",
           "/app/models/worksheet_#{upload.name}.rb",
           "/app/controllers/#{upload.name}_controller.rb"]
  # Delete files
  files.each do |file|
    if File.exist? "#{RAILS_ROOT}/#{file}"
      File.delete "#{RAILS_ROOT}/#{file}"
      puts "deleted #{file}"
    else
      puts "non-existent #{file}"
    end
  end
end

```ruby
def self.remove_mini_app_entry_upload
  # Last, but certainly not least, insert into the MiniApp table
  miniapp = MiniApp.find_by_name upload.name
  if miniapp
    miniapp.destroy
    puts "destroyed #{upload.app} #{upload.version} in the MiniApp table"
  else
    puts "non-existent #{upload.app} #{upload.version} in the MiniApp table"
  end
```
# Create new migrations that will drop the appropriate mantevo result table

def self.create_delete_migrations upload

time = Time.now.utc.strftime "%Y%m%d%H%M%S"

filename = "#{time}_delete_#{upload.name.pluralize}.rb"
text = self.generate_delete_migration upload.name
File.open("#{RAILS_ROOT}/db/migrate/#{filename}", 'w') { |f| f.write text }
puts "created #{RAILS_ROOT}/db/migrate/#{filename}"

filename = "#{time.succ}_delete_worksheet_#{upload.name.pluralize}.rb"
text = self.generate_delete_migration "worksheet_#{upload.name}"
File.open("#{RAILS_ROOT}/db/migrate/#{filename}", 'w') { |f| f.write text }
puts "created #{RAILS_ROOT}/db/migrate/#{filename}"
end

private

def self.generate_delete_migration name

text = "class Delete#{name.pluralize.camelize} < ActiveRecord::Migration

def self.up
begin
drop_table :#{name.pluralize}
rescue
end
end"

end

public

# Remove every migration containing the mantevo-result name of the upload

def self.cleanup_migrations upload

Find.find("#{RAILS_ROOT}/db/migrate/" + migration) do |migration|
  if FileTest.directory?(migration)
    Find.prune if migration.match /.*\.svn.*/
  else
    if migration.match /.*#{upload.name}.*/
      File.delete migration
      puts "deleted #{migration.gsub(RAILS_ROOT, ' ')}"
    end
  end
end
end

private

# app/controllers/worksheet_controller.rb

def self.edit_worksheet_controller name

text = "\n\n# begin #{name}
"}

text << " worksheet
"}

text << " end
"}

end
def delete_#{name}

worksheet_remove_result

end

# worksheet_#{name}

end

# deletion

app/models/worksheet

# Edits the existing worksheet model to create a new through association with
# the newly created tables. We want to be able to do something like the
# following
# to get the results associated with a worksheet: my_worksheet.hpcg_v1point0s

def self.edit_worksheet_model name
  text = "\n\nbegin #{name}\n\nhas_many :worksheet_#{name}.pluralize\n\nend\n"

end

# app/models/user.rb

# The User model needs to know that it now has many of the new Mini-App

def self.edit_user_model name
  text = "\n\n#{name}\n\nhas_many :#{name}.pluralize\n\nend\n"

end

# db/migrations/...

def self.new_migration upload
  name = upload.name
  flat_yaml = YAML::load(upload.flat_yaml)

  text = "\n\nAUTO-GENERATED. See AdminUploadController private method\nnew_migration\n\n\ndef self.up\n\ncreate_table :#{name}.pluralize do, t, \n\nt.text :original_yaml, :default => ''\n\nflat_yaml.each_key do |key|
  text << "\n  \n  if flat_yaml[key].class == Fixnum
    # Note: integers are stored as floats. Intuition tells me that not many
    # integers will be in the YAML anyway, and for the purpose of displaying data, I don't
    # think
    # it will matter if there are decimal points at the end of integers.
    # The potential problem with making these columns integers is that this
    # upload might submit YAML with an integer val when future submissions
    # through
    # the result upload controller will have floating point values...maybe this
    # isn't a valid concern.
  text << "float\n  elsif flat_yaml[key].class == Float
    text << "float\n  elsif flat_yaml[key].class == String

  end

end

end

end

end
# Why add this new result type to the MiniApp table in the migration?
# Because of development reasons. If you play around with the development
# database you may every once in a while drop the database 'rake db:drop'
# and start anew. If all result migrations have this built in, a simple
# rake db:migrate will repopulate the MiniApp table.

def self.find_by_name('#{name}')
  unless app
    app = MiniApp.new
  end
  app.name = '#{name}"
  app.app = '#{upload.app}"
  app.version = '#{upload.version}"

  app
end

# Using gsub takes care of the problem of having quotation marks in the
# original YAML,
# but what about other special characters that could be problematic. The
# system
# could be more robust. Any problems should theoretically get caught by
# testing.
def self
  app.flat_yaml = "#{upload.flat_yaml.gsub(/\"/, "\\\"")}
  app.alternate_names = "#{upload.alternate_names.gsub(/\"/, "\\\"")}
  app.columns = "#{upload.columns.gsub(/\"/, "\\\"")}

  app.save
  t.timestamps
  t
end

def self
  def down
    drop_table :#{name.pluralize} do |t|
      t.column :worksheet_id, :integer, :null => false
    end
  end
end

def self
  new_worksheet_migration name
  text = "\AUTO-GENERATED. See AdminUploadController private method 'new_worksheet_migration' \n"  text << "class CreateWorksheet#{name.pluralize.camelize} < ActiveRecord::
  Migration
end
  def self.up
    create_table :worksheet #:{name.pluralize} do |t|
      t
      end
  end
end

def self
  new_worksheet_migration name
  text = "\AUTO-GENERATED. See AdminUploadController private method 'new_worksheet_migration' \n"  text << "class CreateWorksheet#{name.pluralize.camelize} < ActiveRecord::
  Migration
end
  def self.up
    create_table :worksheet #:{name.pluralize} do |t|
      t
      end
  end
end

text << "  t.column :#{name}_id, :integer, :null=>false
"
352 text << "  \n"
353 text << "  t.timestamps
"
354 text << "  end\n"
355 text << "  end\n"
356 text << "  \n"
357 text << "  def self.down\n"
358 text << "  drop_table :worksheet_#{name.pluralize}\n"
359 text << "  end\n"
360 text << "end\n"
361
end
362
def self.new_model name
363   text = "  \n
# AUTO-GENERATED. See AdminUploadController private method 'new_model
#
"  
364   text << "  class #\{name.camelize\} < ActiveRecord::Base\n"
365   text << "  belongs_to :user\n"
366   text << "  end\n"
367 end
368
end
369
end
370

# app/models/worksheet_app_version.rb
371 def self.new_worksheet_model name
372   text = "  \n
# AUTO-GENERATED. See AdminUploadController\ private method 'new_migration
#
"
373   text << "  class Worksheet#\{name.camelize\} < ActiveRecord::Base\n"
374   text << "  belongs_to :worksheet\n"
375   text << "  belongs_to :#{name}\n"
376   text << "  \n"
377 # a quick and easy way of removing duplicate entries.
378   text << "  validates_uniqueness_of :worksheet_id, :scope => :#{name}_id\n"
379   text << "  end\n"
380 end
381
382 # Creates a controller for the new Mini-App.
383 # Notice that the code seems hollow. Every method will call a corresponding
384 # method that is inherited from the result controller. The @app and ver
385 # set up in a before filter in the ApplicationController is a key player in
386 # allowing this to happen.
387 def self.new_controller name
388   text = "  \n
# GENERATED CODE. See AdminUploadController\n"
389   text << "  class #\{name.camelize\}Controller < ResultController\n"
390   text << "  end\n"
391 end
392
end
393
end
394
end

E.8.2 Stylesheets
The following files can be found in /public/stylesheets/

Listing 89: Main CSS

body { background-color: #fff; color: #333; }
width: 10em;
padding: 4px;
background-color: white;

#nav li ul {
  position: absolute;
  width: 10em;
  left: -999em;
}

#nav li:hover ul, #nav li.sfhover ul {
  left: auto;
}

#header {
  font-size: small;
  min-height: 9px;
  margin: 0;
  padding: 0;
  border-bottom: 1px dotted #aaa;
  background-color: #ddd;
}

#header span {
  color: #666;
  margin: none;
}

#header span#header-left {
  position: absolute;
  top:0px;
  left: 0px;
  padding: none;
  padding-left:4px;
}

#header span#header-right {
  position: absolute;
  top:0px;
  right:0px;
  padding: none;
  padding-right:8px;
}

#user-edit{
  color: red;
}

#header span a {
  text-decoration: none;
.result-table a.users-to-nu:hover {
  color: #64a6c4;
}

.result-table tr.line-odd{
  background-color: white;
  color: #4f4f4f;
}

.result-table tr.line-even {
  background-color: #cccccc;
  color: #4f4f4f;
}

.result-table h1 {
  font: 150% serif;
  color: #64a6c4;
  border-bottom: 3px dotted #64a6c4;
  padding-left: 0.5em;
}

.chart-and-table {
  clear:both;
  margin: 20px;
  position: relative;
}

.chart-and-table .chart {
  float:left;
}

.chart-and-table .result-table {
  float:left;
}

#detail-header{
  color: #4f4f4f;
}

#detail-header h4{
  background: #555555;
  padding-left: 4px;
  color: #eeeeee;
  display: inline;
}

#detail-header h4.menu{
  background: #888888;
  display: inline;
}

#detail-header h4.menu:hover, #detail-header h4.menu-active:hover{
  background: #999999;
}

#detail-header h4.menu-active{
  background: #4f4f4f;
}
# detail-header h4 a{
  color: #e6e6e6;
  text-decoration: none;
}

# detail-header h4 a:hover{
  color: #fff;
}

# detail-header h5{
  display:inline;
  background: white;
  color: #666666;
  padding-left: 4px;
  margin-right: 50px;
}

# detail-body{
  padding: 8px;
  background: #444444;
  height: auto;
}

# detail-body .charts{
  margin: 8px;
}

# detail-body .result-table{
  margin: 8px;
}

# detail-body #yaml{
  margin: 8px;
}
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YAML vs. XML</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>HTML Table (standard view)</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>HTML Table (sorted on a column)</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>HTML Table (logged in user view)</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>HTML Table (worksheet view)</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>Detail View (original YAML)</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>Detail View (flat table)</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>Detail View (charts)</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>A Worksheet with a Chart</td>
<td>22</td>
</tr>
<tr>
<td>10</td>
<td>Admin Upload New Result YAML</td>
<td>24</td>
</tr>
<tr>
<td>11</td>
<td>Altering Names</td>
<td>24</td>
</tr>
<tr>
<td>12</td>
<td>Selecting Columns for HTML Table Views</td>
<td>25</td>
</tr>
<tr>
<td>13</td>
<td>Core Database</td>
<td>26</td>
</tr>
<tr>
<td>14</td>
<td>ResultType</td>
<td>27</td>
</tr>
<tr>
<td>15</td>
<td>Admin</td>
<td>27</td>
</tr>
<tr>
<td>16</td>
<td>Alternate Names Example</td>
<td>29</td>
</tr>
<tr>
<td>17</td>
<td>Columns Example</td>
<td>29</td>
</tr>
<tr>
<td>18</td>
<td>MVC Architecture (image taken from Thomas, p 23)</td>
<td>32</td>
</tr>
</tbody>
</table>

## Listings

<table>
<thead>
<tr>
<th>Listing</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YAML</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>XML</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Original</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>Flattened</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>SJU 1.0</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>SJU 2.0</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>SJU 2.0 Could Look Like This</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>SJU 2.0 Could Look Like This</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>Alternate Names Example</td>
<td>29</td>
</tr>
<tr>
<td>10</td>
<td>Columns Example</td>
<td>29</td>
</tr>
<tr>
<td>11</td>
<td>SJU 0.8</td>
<td>39</td>
</tr>
<tr>
<td>12</td>
<td>SJU 0.9</td>
<td>39</td>
</tr>
<tr>
<td>13</td>
<td>Workout - 3 sets</td>
<td>41</td>
</tr>
<tr>
<td>14</td>
<td>Workout - 4 sets</td>
<td>41</td>
</tr>
<tr>
<td>15</td>
<td>Workout - N sets</td>
<td>42</td>
</tr>
<tr>
<td>16</td>
<td>YAML Doc</td>
<td>58</td>
</tr>
<tr>
<td>17</td>
<td>YAML Doc header</td>
<td>39</td>
</tr>
<tr>
<td>18</td>
<td>YAML Element</td>
<td>61</td>
</tr>
<tr>
<td>19</td>
<td>YAML Element header</td>
<td>63</td>
</tr>
<tr>
<td>20</td>
<td>Gather YAML</td>
<td>65</td>
</tr>
<tr>
<td>21</td>
<td>Admin Login Controller</td>
<td>68</td>
</tr>
<tr>
<td>22</td>
<td>Admins Controller</td>
<td>68</td>
</tr>
<tr>
<td>Page</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>23</td>
<td>Admin Upload Controller</td>
<td>70</td>
</tr>
<tr>
<td>24</td>
<td>Application</td>
<td>74</td>
</tr>
<tr>
<td>25</td>
<td>Login Controller</td>
<td>75</td>
</tr>
<tr>
<td>26</td>
<td>Result Controller</td>
<td>76</td>
</tr>
<tr>
<td>27</td>
<td>Result Upload Controller</td>
<td>84</td>
</tr>
<tr>
<td>28</td>
<td>User Controller</td>
<td>85</td>
</tr>
<tr>
<td>29</td>
<td>Worksheet Controller</td>
<td>87</td>
</tr>
<tr>
<td>30</td>
<td>Application Helper</td>
<td>89</td>
</tr>
<tr>
<td>31</td>
<td>Result Helper</td>
<td>90</td>
</tr>
<tr>
<td>32</td>
<td>Worksheet Helper</td>
<td>91</td>
</tr>
<tr>
<td>33</td>
<td>Admin</td>
<td>92</td>
</tr>
<tr>
<td>34</td>
<td>Detail Chart</td>
<td>93</td>
</tr>
<tr>
<td>35</td>
<td>MiniApp</td>
<td>93</td>
</tr>
<tr>
<td>36</td>
<td>Upload</td>
<td>94</td>
</tr>
<tr>
<td>37</td>
<td>User</td>
<td>94</td>
</tr>
<tr>
<td>38</td>
<td>Worksheet</td>
<td>95</td>
</tr>
<tr>
<td>39</td>
<td>Worksheet Chart</td>
<td>95</td>
</tr>
<tr>
<td>40</td>
<td>Admin Login Views - Index</td>
<td>96</td>
</tr>
<tr>
<td>41</td>
<td>Admins Views - Edit</td>
<td>96</td>
</tr>
<tr>
<td>42</td>
<td>Admins Views - Index</td>
<td>96</td>
</tr>
<tr>
<td>43</td>
<td>Admins Views - New</td>
<td>97</td>
</tr>
<tr>
<td>44</td>
<td>Admin Upload Views - Column Selection</td>
<td>97</td>
</tr>
<tr>
<td>45</td>
<td>Admin Upload Views - Edit Column Names</td>
<td>98</td>
</tr>
<tr>
<td>46</td>
<td>Admin Upload Views - Finish Upload</td>
<td>99</td>
</tr>
<tr>
<td>47</td>
<td>Admin Upload Views - Index</td>
<td>99</td>
</tr>
<tr>
<td>48</td>
<td>Admin Upload Views - Review</td>
<td>100</td>
</tr>
<tr>
<td>49</td>
<td>Layouts Views - Admins</td>
<td>101</td>
</tr>
<tr>
<td>50</td>
<td>Layouts Views - Header partial</td>
<td>101</td>
</tr>
<tr>
<td>51</td>
<td>Layouts Views - Main</td>
<td>102</td>
</tr>
<tr>
<td>52</td>
<td>Layouts Views - Main Menu partial</td>
<td>102</td>
</tr>
<tr>
<td>53</td>
<td>Layouts Views - Meun Javascript partial</td>
<td>103</td>
</tr>
<tr>
<td>54</td>
<td>Login Views - Index</td>
<td>103</td>
</tr>
<tr>
<td>55</td>
<td>Login Views - Register</td>
<td>103</td>
</tr>
<tr>
<td>56</td>
<td>Result Views - Create Detail Charts partial</td>
<td>104</td>
</tr>
<tr>
<td>57</td>
<td>Result Views - Create Worksheet Charts partial</td>
<td>104</td>
</tr>
<tr>
<td>58</td>
<td>Result Views - Detail partial</td>
<td>105</td>
</tr>
<tr>
<td>59</td>
<td>Result Views - Edit Detail Chart partial</td>
<td>106</td>
</tr>
<tr>
<td>60</td>
<td>Result Views - Edit Worksheet Chart partial</td>
<td>107</td>
</tr>
<tr>
<td>61</td>
<td>Result Views - Header partial</td>
<td>108</td>
</tr>
<tr>
<td>62</td>
<td>Result Views - Home</td>
<td>108</td>
</tr>
<tr>
<td>63</td>
<td>Result Views - Info partial</td>
<td>108</td>
</tr>
<tr>
<td>64</td>
<td>Result Views - Results partial</td>
<td>108</td>
</tr>
<tr>
<td>65</td>
<td>Result Views - Standard partial</td>
<td>109</td>
</tr>
<tr>
<td>66</td>
<td>Result Views - Standard User partial</td>
<td>109</td>
</tr>
<tr>
<td>67</td>
<td>Result Views - Worksheet partial</td>
<td>110</td>
</tr>
<tr>
<td>68</td>
<td>Result Upload Views - Index</td>
<td>111</td>
</tr>
<tr>
<td>Page</td>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>69</td>
<td>User Views - Edit</td>
<td>111</td>
</tr>
<tr>
<td>70</td>
<td>User Views - Index</td>
<td>111</td>
</tr>
<tr>
<td>71</td>
<td>User Views - New Worksheet</td>
<td>112</td>
</tr>
<tr>
<td>72</td>
<td>User Views - Users partial</td>
<td>112</td>
</tr>
<tr>
<td>73</td>
<td>User Views - Worksheets partial</td>
<td>112</td>
</tr>
<tr>
<td>74</td>
<td>Worksheet Views - Index</td>
<td>113</td>
</tr>
<tr>
<td>75</td>
<td>Worksheet Views - Worksheet partial</td>
<td>113</td>
</tr>
<tr>
<td>76</td>
<td>Worksheet Views - Worksheet Detail</td>
<td>113</td>
</tr>
<tr>
<td>77</td>
<td>Result.Rake</td>
<td>114</td>
</tr>
<tr>
<td>78</td>
<td>Database YAML</td>
<td>117</td>
</tr>
<tr>
<td>79</td>
<td>Create Users</td>
<td>117</td>
</tr>
<tr>
<td>80</td>
<td>Create Worksheets</td>
<td>117</td>
</tr>
<tr>
<td>81</td>
<td>Create Admins</td>
<td>118</td>
</tr>
<tr>
<td>82</td>
<td>Create MiniApps</td>
<td>118</td>
</tr>
<tr>
<td>83</td>
<td>Create Sessions</td>
<td>118</td>
</tr>
<tr>
<td>84</td>
<td>Create Uploads</td>
<td>119</td>
</tr>
<tr>
<td>85</td>
<td>Create Detail Charts</td>
<td>119</td>
</tr>
<tr>
<td>86</td>
<td>Create Worksheet Charts</td>
<td>120</td>
</tr>
<tr>
<td>87</td>
<td>Hash Methods</td>
<td>120</td>
</tr>
<tr>
<td>88</td>
<td>Result Helper</td>
<td>121</td>
</tr>
<tr>
<td>89</td>
<td>Main CSS</td>
<td>128</td>
</tr>
</tbody>
</table>