Design, Development, and Implementation of an Information Architecture for OIT Health and Informatics SIMMLAB

A Senior Project Submitted to the Department of Management, Oregon Institute of Technology in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science

John Fox

June 5, 2014

Table of Contents

Executive Summary 4

Project Overview 5

Problem Definition 5

Problem Significance 5

Major Milestones/Deliverables 5

Project Requirements 6

Planning Methodology 6

Requirements Overview 6

Design Influence of Functional and Nonfunctional Requirements 6

Non-Functional Specification 7

Summary 7

Design Description 9

Requirements-to-Features 9

Requirement: Standardized Documentation 9

Requirement: Categorized Searchable Knowledge Base with Procedure Moderation 9

Requirement: Event Log 10

Brief Discussion of User-Interface Design 10

Project Implementation 11

Overview 11

Test Plans and Beta-Testing 12

Installation of Final Project 12

Install Complementary Software 12

Kable Setup 13

Manuals and Other Documentation 16

Summary 17

Project Summary 17

Future Direction 17

Lessons Learned 17

Appendix A – Project Proposal 18

Executive Summary 18

Project Overview 18

Project Description and Objectives 18

Stakeholders 19

Major Goals 19

Planning and Analysis 19

Alternate Methods 20

Project Scope 20

Outside Scope 21

External Inputs 21

Overview of Solution 21

High-Level of Solution 21

Major Features, Functions, and Deliverables 22

Chosen Project Management Life Cycle 23

Issues to Resolve 24

Addendum 24

Project Overview Statement 24

Problem/Opportunity 24

Goal 25

Objectives 25

Success Criteria 26

Assumptions, Risks, Obstacles 26

Appendix B – Testing 27

Field Validation 27

Common Testing of Application During Development 27

Appendix C – Kable User’s Manual 28

About Kable 28

Interface Conventions and Application Menu Tour 28

Administrative Functions 29

User Functions (Events and Procedures) 30

Templates and Style Guide 32

Slainte! 32

# **Executive Summary**

The project goal is the design and implementation of an information architecture for the Oregon Institute of Technology's Health and Informatics program's computer lab environment (SIMMLAB). The project has three global requirements:

1. Standardized documentation with policies linked to procedures, and style guide for procedure write-ups.
2. Event log
3. Categorized searchable knowledge base with moderation for new and edited procedures.

These requirements are fulfilled by a single system built in ASP.NET and C#.

# Project Overview

## Problem Definition

OIT's Wilsonville campus hosts the Health and Informatics SIMMLAB--a cloud-computing environment used to provide HI students exposure to the HI software they'll be using in the real world. Students administer the SIMMLAB, many of whom know nothing of system administration when beginning their term of service. These administrators need a guide that will list problems as they come up, and store solutions for future reference.

## Problem Significance

The SIMMLAB software is very expensive, with expensive support contracts. The school's budget allocates very few dollars to this resource. The instructors overseeing the lab have thus had to ponder how the lab might sustain itself, and thus remain in place as a resource for the OIT program.

One possibility is to sell SIMMLAB access to other schools that have HI programs, but lack a good lab environment. Should this course be taken, it's even more essential that the lab's student administrators have informational resources that will promote swift, effective administration.

## Major Milestones/Deliverables

Project deliverables fall into two categories--documentation and application. The documentation milestones were a policy template, procedure templates, and a style guide governing documentation of procedures. The application milestones were the application's "administration" suite (intended for system administrators), "event" suite (intended for users to document and view events and procedures), and the templates suite (provides access to online version of the policy template, procedure template, and style guide.

# Project Requirements

## Planning Methodology

I spoke with four stakeholders in the planning phase of this project. OIT's Jeff Dickson stated that he felt an event log was vital. John Stephens, of Luminant Digital Security, spoke of the need for standardized documents, and ensuring that all procedures are associated with a policy, and using templates for policies and procedures to aid in document standardization. Huron Consult's Phil Wyckoff wanted a "recipe book" for lab administrators. From Grant Kirby came requests for a searchable knowledge base with moderator approval required for new procedures. Mr. Kirby also requested a style guide governing procedure write-ups.

In preparation, I checked other OIT systems to see how their information was categorized. Specifically, I checked the "One Note" knowledge base, and "FACTS", the ticketing system used by the IT department. These investigations gave me the first concrete result of my planning: they aided me in determining the categories I would later use for classifying events and procedures. This gave me my first sense of progress in the project, and it was around this time that I had to determine which SDLC approach I would use for the project.

I had good information about the project's goals and solutions. I also had very low likelihood of outside factors impacting my project. Because of these factors, I chose the "Linear" project management lifecycle. I was glad that these conditions applied, because I felt that this approach had a bit less uncertainty, and I wanted to minimize uncertainty in my first big solo project.

## Requirements Overview

Based on the stakeholder interviews, I came up with three global requirements. Each requirement was broken down into atomic units, with each unit representing a feature. These global requirements are:

1. Standardized documentation with policies linked to procedures, and style guide for procedure write-ups.
2. Event log
3. Categorized searchable knowledge base with moderation for new and edited procedures.

In the next section, I discuss these requirements, and the features derived from them.

## Design Influence of Functional and Nonfunctional Requirements

I decomposed the following features from the "standardized documentation..." requirement:

* Policy template
* Procedure template
* Ability to create and edit policies
* Ability to remove policies not linked to a procedure
* Ability to edit existing procedures
* Ability to create procedure and tie to a policy
* Style guide for procedure write-ups.

From the "event log" requirement, I derived the following features:

* Ability to document events
* Ability to tie event to information taxonomy category
* Information taxonomy (categories, e.g. "VPN", "Firewall")

The "categorized searchable knowledge base..." requirement decomposed into the following features:

* Per-category procedure search
* Per-policy procedure search
* Email alert to moderator upon creation or alteration of a procedure
* Ability for moderator to approve procedure in moderation
* Information taxonomy (categories for procedures and events)

## Non-Functional Specification

Many of these features require an information store. This project makes use of Microsoft's "SQL Server 2012 Express" product for information storage and manipulation.

The application is intended to have access control, ensuring that regular users cannot make use of the system's administrative features. My plan called for using Windows domain users and groups to provide that access control.

The application was developed using Windows 7 and ASP.NET, and is intended for deployment under IIS, Microsoft’s web server product. I am not particularly fond of Windows tools, but made this choice because in this case, they seemed the best for the job. This judgment was based on two primary factors:

1. The SIMMLAB is a Windows environment, and OIT teaches the Windows tools; it seemed unfair to future administrators to give them a non-Windows solution[[1]](#footnote-1).
2. I had previous classroom experience with creating database-backed Web applications using ASP.NET and C#.

## Summary

My original planning was based on discernment of requirements from stakeholder interviews and on a study of the information taxonomies used by other IT tools in use at OIT. Based on the facts that my project’s goals and solutions seemed clear, and a low likelihood of external factors impacting my plans, I chose to develop this project under the *linear* project development lifecycle.

The system’s global requirements were:

* Standardized documentation, with policies linked to procedures, and a style guide for procedural write-ups
* Categorized, searchable knowledge base of procedures
* Event log for documenting trouble events

My primary nonfunctional requirements were a relational database management system, Windows Active Directory-based access control, and a web-based application written with—and deployed to—Windows systems.

After establishing requirements, I moved on to the application’s design.

# Design Description

In this section I will 1) explain the thought processes that governed my decomposition of the requirements into individual features, and 2) touch briefly on the application’s user-interface design.

## Requirements-to-Features

### Requirement: Standardized Documentation

Stakeholder John Stephens emphasized the importance of policies and procedures. He stated that each procedure should be associated with a relevant policy, and that the writing of policies and procedures should be based on templates to ensure standardization. Along similar lines, stakeholder Grant Kirby felt it important that procedural write-ups should be written in accordance with a style guide. Table 1 displays the features associated with this requirement, and explains each feature’s derivation from the requirement.

|  |  |
| --- | --- |
| **Feature** | **Derivation** |
| Policy template | Direct request |
| Procedure template | Direct request |
| Style guide for procedural write-ups | Direct request |
| Ability to create a procedure and associate it with a policy | The procedure-policy association was a direct request, and implies a need for ability to create a procedure |
| Ability to remove policies unassociated with procedures. | Sometimes, policies change. There is no point in storing an obsolete policy, making it necessary to provide a means of removing policies. |
| Ability to edit existing procedures | People make mistakes. The application must provide users a way to correct errors. |
| Ability to create policies | Given that the application requires a procedure be linked with a policy, it’s a given that the application needs to display policy content. This implies a need to provide a means of adding policies to the system. |

Table 1: “Standardized documentation” features and their derivation from the requirement.

### Requirement: Categorized Searchable Knowledge Base with Procedure Moderation

Stakeholder Grant Kirby wanted non-keyword search functionality for locating procedures. He also requested an administrative moderation feature for new procedures, so that an administrative user could check new entries for correctness. Table 2 shows the features derived from this requirement.

|  |  |
| --- | --- |
| **Feature** | **Derivation** |
| Per-category procedure search | This provides an alternative to keyword search. |
| Per-policy procedure search | This provides an alternative to keyword search. |
| Email alert to moderator on creation or editing of a procedure | Direct request |
| Moderation of new or edited procedures | Direct request. |

Table 2: “Categorized Searchable Knowledge Base….” features and their derivation from the requirement.

### Requirement: Event Log

Stakeholders Grant Kirby and Jeff Dickson wanted an event log. This log would serve as a record of all problem events encountered by SIMMLAB administrators. See Table 3 for a listing of the features derived from this requirement and an explanation of how these features relate to the requirement.

|  |  |
| --- | --- |
| **Feature** | **Derivation** |
| Ability to document events | Direct request |
| Ability to associate an event with a relevant information taxonomy category (e.g. “VPN” or “Firewall”) | It seemed to me that it would be bad practice to lump all events into a single bucket. I therefore chose to enforce an event-category association. |
| Ability to add and remove event categories | The system is inflexible if it’s loaded with a static information taxonomy, so it seemed appropriate to allow for editing, creation, and removal of categories. |
| Ability to browse events by category | Given that I had decided each event would be associated with a category, it seemed only sensible to allow the event listing to be sorted by category. |

Table 3: “Event Log” requirement features, and their rationale.

## Brief Discussion of User-Interface Design

I wanted the user interface to be very consistent. Therefore, all creation, editing, and deletion features have the same interface, whether they pertain to policies, procedures, categories, events, application configuration, or procedure moderation. Figure 1 illustrates the basic layout of these interfaces.

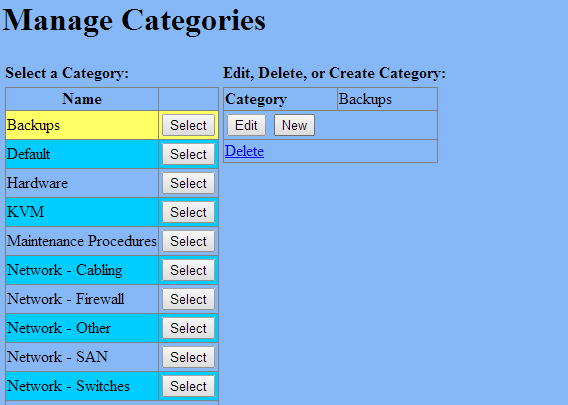


Figure : Example interface

In each interface, a gridview control on the left displays a listing of relevant items (event and procedure categories in this case). An explanatory caption above the gridview instructs the user to make a selection from the list, and a caption to its right explains what can be done once a selection has been made. When a selection is made, its data appears in a dataview control. At the bottom of the dataview control are buttons for operating on the selected item, or creating a new item. If the user wishes to delete the selected item, she clicks the “Delete” linkbutton and answers “yes” to the confirmation prompt that appears. If she wishes to edit the selected item, she clicks the “Edit” button, makes her changes, and confirms the edit by clicking the “Update” button that appears. If the user wishes to create a new item, she clicks the “New” button, enters the requested data, and confirms the new data by clicking the “Insert” button that appears.

That completes the “Design” portion of this document. In the next section, I will discuss the project’s “implementation” phase.

# Project Implementation

## Overview

In a footnote at the end of the “Project Requirements” section, I alluded to a design change, and stated that it would be explained in the “Implementation” section. The time for that explanation has now arrived.

The project has seen one major change from the original plans, and it occurred while I was implementing the application. Because I come from a UNIX tradition rather than a Windows one, my original intent was to have the knowledge base and event log as two separate systems running in Linux environments, with a loose integration between the two. I had planned to use the open source “Request Tracker” ticketing system as the event log, and—because I could not find a ready-made Linux knowledge base software package—I intended to develop the knowledge base from scratch, using the Ruby on Rails[[2]](#footnote-2) web application framework. Both systems would use a MySQL database for information storage.

I encountered no difficulties implementing the “Request Tracker” component. Or, perhaps I should say, no show-stopping difficulties. In any event, that portion of the system was completed in a relatively timely fashion. However, when I began to implement the knowledge base, I soon realized that Rails development was much more involved than I’d realized. I knew that I could learn it, but I also knew that I could not do so in time to meet the project deadline; I had to iterate, and think of another approach.

After a little thought, I realized that my ASP.NET, C#, and SQL Server experience from previous classes could be applied to the project. ASP.NET, I had read, was well suited to rapid application development, and my classroom experiences seemed to confirm this. On the heels of this realization came another; I could use these tools to create the event log as well. This would provide a much more consistent user experience—a net gain for both the system’s eventual users and me.

As I began my ASP.NET implementation, I tried to figure out an approach that would allow me to test my application as I was developing it. I believe that I was successful.

## Test Plans and Beta-Testing

I had concerns about testing—the application was coming along surely, but slowly as well. I encountered many small concerns and issues that hadn’t come up in my classes. Finally, I resolved that with the relatively simple system I was creating, the greatest chance for error came from allowing bad data into the database. I would, therefore, make extensive use of ASP.NET’s validation controls, which allow data-entry fields to be tested for values to be present and for specific data types to be required. For further details on this approach, see Appendix B.

## Installation of Final Project

The project is designed to be managed and hosted from a single Windows 7 host, which must be running a Windows 7 release that supports joining Active Directory domains. In my development, I used Windows 7 Ultimate Edition. A stock Windows 7 Ultimate installation, however, does not suffice for the project. There are a few other packages that must be installed.

## Install Complementary Software

**SQL Server 2012 Express**

SQL Server 2012 Express can be freely downloaded from Microsoft. It should be installed on the Windows 7 host. When configuring the initial instance, name it “kable”—this will simplify things a bit later in the installation process. At some point in the installation, the installer will ask what sort of authentication should be used for database connections—**do not** choose “Windows Authentication”. Take the other option instead; this will cause the system to default to mixed-mode authentication.

**Visual Studio Express 2012 for Web**

This product, too, may be downloaded from Microsoft’s web site. Take all default installation values.

**. NET Framework 4.5.1**

The application requires that this version of the .NET framework be present on the host machine. It, too, may be freely downloaded from Microsoft’s site. As with VSE 2012 for Web, there are no installation options to worry over.

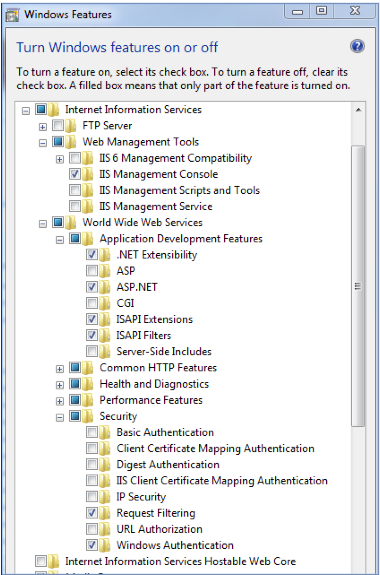


Figure : IIS Features

**Internet Information Service 7.5**

Install IIS by going to the “Programs and Features” control panel and selecting the “Turn Windows features on or off” link. Enable the options shown in Figure 2, and click OK.

## Kable Setup

Kable (a play on KBEL, an acronym for “Knowledge Base/Event Log”) requires some set up. However, I am not able to provide full details on this, because I was never able to make Windows authentication work with the IIS in controlling access to the application[[3]](#footnote-3). I will, however, provide what clews I am able.

Note: These instructions assume that the user is logged in as an administrator.

The DVD in the project binder contains a “Database” folder. In it are two files: kbel.mdf and kbel\_log.ldf. Copy these files to the SQL Server database directory (C:\Program Files\Microsoft SQL Server\MSSQL11.KABLE\MSSQL\DATA). Start the SQL Server Management Studio and attach the kbel database. Create a login named “kable”, and then create a user based on that login. Grant that user read, update, delete, and insert privileges on the following kbel tables: Categories, Config, Events, Policies, and Procedures.

From the project DVD’s “Application Source” folder, copy the “kable” folder to any location on drive C. Enter the folder, and double-click the “kable.sln” file to load the website in Visual Studio Express 2012 for web.

In Visual Studio, open the Solution Explorer. Double-click on “web.config” in the listing. The “web.config” file is displayed. Find the “ConnectionStrings” area, comment out the existing connection string, and add in one that makes use of the “kbel” SQL Server user created earlier. It will look something like this:

*<add name="kbelConnectionString" connectionString="Data Source=oswin\kable;Initial Catalog=kbel;User ID=kable;Password=Kable627!"*

*providerName="System.Data.SqlClient" />*

Replace “oswin” with the short hostname of the host computer, and the “Password” value with the password assigned to the “kable” SQL Server user created earlier. Use Ctrl-S to save your change.

Use Windows Explorer to rename C:\inetpub\wwwroot to C:\inetpub\kable-site.

Return to Visual Studio, and select “Build Web Site” from the “Website” menu. When the build completes successfully, select “Copy Web Site” from the “Website” menu. A new window opens, showing the Kable source directory in the left pane. The right pane contains a blank area labeled “Remote Web site”. At the top of this Window is a “Connect” button. Select it, and a file system navigation box appears. Select the “File System” button in the upper left corner, and navigate the local directory tree, ultimately selecting the folder “C:\inetpub\kable-site”, and click the “Open” button in the window’s bottom right corner. Single-click any file in the “Source Web site” listing, and then issue a Ctrl-A command to select all files. Click the “->” button between the two panes to copy the website files into the destination directory. You are now done with Visual Studio, and may shut it down.

Click the Start button and type “IIS Manager”—“internet Information Services (IIS) Manager” appears at the top of the start menu window. Select it to run the IIS Manager. When the IIS Manager has loaded, fully expand the tree under the “Connection” pane on the left side. Right-click “Default Web Site” and choose to remove it.

Now add the kable site. In the connection tree, right-click the “Sites” icon, and choose “Add Web   
Site”. A dialog box appears. Enter “kable” in the site name field. It will also appear in the “Application pool” field. To the left of that field is a “Select…” button. Click it, and then choose “ASP.NET v4.0” from the list of application pools, and click “OK”.

To the right of the “Physical path” field is a “…” button. Select it to launch a folder browser. Use it to select C:\inetpub\kable-site.

Further down, is a field labeled “hostname”. Here, enter the computer’s fully qualified hostname.

Ensure that the “Start Web Site immediately” box is checked, and click “OK”. IIS then starts the website, and the IIS Manager interface appears again. Select the “kable” website. To the left of the “Connection” pane is a series of icons, some under the “ASP.NET” heading, and others under the “IIS” heading. From the IIS area, select “Authentication”. The only method that should be enabled is “Windows Authentication”. Enable by right-clicking on it and selecting “Enable”. Disable the others by right-clicking on them and selecting “Disable.”

This is as far as I can go in detailing the Kable set-up procedure. It is here that I ran aground while trying to work out how to make IIS work with Windows authentication.

# Manuals and Other Documentation

The Kable user manual is divided into three parts. The first explains the general interface conventions for creating items, deleting items, editing items, modifying application configuration, and approving procedures in moderation. This material appeared earlier in this document, under the “Design Description” section. The second manual section details the application’s administrative functions, and the third explains the user functions related to documenting events and procedures, and browsing procedures sorted by various criteria.

It is expected that the Kable administrator will be an experienced computer user, but the material within the administrative portion will not be particularly complex. Nor will that in the user portion of the manual, because of the very real possibility that the regular users will be new to such a system. However, in deference to possible inexperience, the user portion of the manual will supplement its textual explanations with graphics.

The user manual appears in Appendix C.

# Summary

## Project Summary

The original goal was to achieve a loose integration between two systems, a pre-package ticketing system to serve as an event log, and a custom –built Ruby On Rails application to provide the knowledge base functionality. This would not have been an ideal approach, primarily because users would have had to learn two different interfaces, and the cross-system integration would likely have provided an unsatisfying experience. In fact, for all I know, certain aspects of that integration might have posed significant technical hurdles.

So it is probably better all around that I had to iterate and find a different approach. I ended up with a single system that provided a consistent user interface, and which I was able to complete prior to the project’s deadline.

## Future Direction

I have a few thoughts for additional functionality. A future student could add in a linkage between a procedure and the event that inspired it. Perhaps even keyword search could be added one day—it might be that some people would prefer that approach.

## Lessons Learned

As I sit here, typing this, having completed the project as best I could in the allotted time, one concept springs to mind: complexity. Going into this project, I knew intellectually that even a simple system is fairly complex, but I hadn’t fully internalized that idea. It wasn’t until I began my implementation and hit a million unexpected snags that I fully realized this truth. From that realization come the two primary lessons I learned in the course of the project.

Time management and planning are vital. With better use of time, perhaps I would have been able to get Windows authentication working. If I’d planned better, instead of simply assuming that the IIS configuration for Windows authentication would be simple, perhaps I would have had less stress in these final few weeks.

Sometimes iteration is necessary. There are points in a project’s development where plans go awry, and while persistence is often a virtue, it is important to recognize that sometimes it’s better to iterate, and try another approach. If I had refused to recognize that my Ruby on Rails approach was more than I could handle, how many more hours would I have poured into it? Perhaps I would have eventually realized I had to try something different, but would I have done so in time to complete the project using ASP.NET, C#, and SQL Server?

This project caused me a lot of stress. It was like nothing I’d ever attempted to before, and I spent a fair amount of time feeling frustrated and resentful. But as I look back it all now, I realize I’m proud of the work I’ve done. I’m grateful to have had the experience.

# Appendix A – Project Proposal

This is the project proposal submitted in December of 2013. It was the first of many documents generated in the course of the project, and is included here primarily to show what aspects of the project remained unchanged, and which did not.

## Executive Summary

This project's goal is to design and implement an information infrastructure for the OIT Health and Informatics SIMMLAB. It has the following global requirements:

• Standardized documentation by way of templates for policies and procedures, creation of a style guide for procedures, and enforcing a linkage between policies and procedures.

• A knowledge base for storage of policies and procedures.

• Moderation for new procedures, with an alert to moderator(s) when a new procedure has been submitted.

• A searchable ticketing system to serve as event log.

• Cross-system integration between knowledge base and event log, allowing single sign-on.

• Incorporation of content from current SIMMLAB knowledge base, and information generated by Mark Miller during his work documenting the SIMMLAB "as-is" system.

The knowledge base will be custom-built using a pre-existing web application framework (Ruby on Rails) and the event log will be a slightly customized version of the "Request Tracker" ticketing system.

## Project Overview

### Project Description and Objectives

The OIT Health and Informatics program uses the SIMLABB to train students in the software they will encounter in the real world. Licensing and support for the lab's software and hardware is expensive, and the school is unable to renew support contracts. SIMMLAB stakeholders hope to offset this lack of funding by renting the lab's services to other schools running Health and Informatics programs.

Although the SIMMLAB is--and has been--run by knowledgeable, competent individuals, there are many changes to be made before renting the environment to other schools is feasible. Among these are full documentation of the existing system's procedures and configurations, and creation of an information architecture designed to incorporate this (and future) documentation in a manner that lends itself to search by way of information categorization.

### Stakeholders

This project's key stakeholders are OIT instructors Grant Kirby and Jeff Dickson; commercializing the SIMLABB is their vision. Other stakeholders are John Stephens of Luminant Digital Security, Phil Wyckoff (a former OIT student who had a huge part in defining the SIMLABB's current configuration), and Mark Miller (a current OIT student whose senior project is documenting the "as-is" SIMMLAB.

### Major Goals

The project has four major goals:

• Simple navigation of technical material

• System will be able to accommodate growth

• Style guide and templates for knowledge base entries

• Incorporate contents of both current knowledge base and information generated by Mark Miller's documentation of "as-is" system.

## Planning and Analysis

### Alternate Methods

Of necessity, any software chosen for this project must be open source (i.e. no licensing charges, and source code freely available.

While researching possibilities for the ticketing system that will serve as event log, I discovered several open source applications, among them OTRS ("Open source Ticket Request System"), "osTicket," "Simple Ticket," "eTicket," "Trouble Ticket Express," "itracker," "Jutda Helpdesk," and "Mystic" (OSH Editorial).

I did not find any existing software application that could provide the requisite policy-procedure linkage. The only choices were to create my own or alter an existing "web log" package. I had two reasons for choosing to create my own:

1) I felt that adding the required changes to an existing, complicated system might lead to a complexity I could not overcome, whereas if I created the system myself, I would understand it quite well.

2) Once I got the idea of creating the application myself, I realized that I was more interested in that course of action than the other.

### Project Scope

This project's scope encompasses:

• Cross-system navigation between knowledge base and event log, providing single sign-on and hyperlinks allowing cross-category navigation between the two systems.

• Incorporation of current knowledge base articles and those generated by Mark Miller in his project.

• Drill-down style search of knowledge base by category and policy.

• Installation and configuration of a ticketing system to serve as an event log.

• Moderation function and alerts when a new procedure has been submitted.

### Outside Scope

The knowledge base will not support keyword searching; Grant Kirby specifically mentioned wanting a "drill down" approach, and I felt that adding keyword search to feature list might contribute to a time deficit.

### External Inputs

This project's external inputs are related to Mark Miller's project; he and I will decide upon the information categories to be used, and his documentation of the "as-is" system will be incorporated into knowledge base content.

## Overview of Solution

### High-Level of Solution

#### Event Log

As indicated in "Alternate Methods," I discovered several open source ticketing systems. In the end, I chose "Request Tracker" based on the following factors:

• It is widely used in the enterprise ("Who uses rt?,”)

• In a previous, tech support job, we used RT as our ticketing system.

• It's written in Perl, a language I'm familiar with, which I expect will simplify my customization.

#### Knowledge Base

The knowledge base will created specifically for this project. I did not wish the added complexity of creating the entire application from scratch, so it will be built on top of the "Ruby on Rails" web application framework.

These systems will share user authentication credentials, and provide category-based hyperlinks from one system to another (i.e. when viewing a "Network" ticket, a hyperlink to the knowledge base's "Network"-related procedures will be provided).

### Major Features, Functions, and Deliverables

#### Features and Functions

The global requirement "standardized documentation with policies linked to procedures, and style guide" yielded these features and functions:

• Policy template

• Procedure template

• Ability to edit existing policies

• Ability to create new policies

• Ability to delete policies not associated with procedures

• Ability to edit existing procedures

• Ability to create new procedure and associate with a policy

• Ability to delete an existing procedure

• Style guide for procedure write-ups

The global requirement "categorized searchable knowledge base with moderation alert" calls for these features and functions:

• Per-category drill-down procedure search

• Per-policy drill-down procedure search

• Email moderator upon submission of new procedure

• Information taxonomy for procedure categorization (e.g. "Network," "Operating System," etc.)

The "event log" requirement necessitates:

• ticketing system

• custom information taxonomy added to ticketing system

The "cross-system integration" requirement yield these features and functions:

• Synchronize user names and passwords for both software systems

• Deletion of event log user accounts when user removed from knowledge base

• Creation of event log user accounts when user added to knowledge base

• Event log ticket view contains a hyperlink to corresponding category procedures in knowledge base

• Knowledge base procedure view contains a hyperlink to corresponding category tickets in event log

The "incorporate current and Mark Miller-generated content" requirement calls for these functions:

• articles categorized

• articles associated with relevant policies

• articles submitted to knowledge base

#### Deliverables

These deliverables are taken from the project timeline, which can be found in the addendum:

• style guide

• policy template

• procedure template

• information taxonomy

• Request Tracker ticketing system with custom information taxonomy configured

• base knowledge base application

\* cross-system integration of user accounts and navigational categories

• content of "as is" knowledge base system incorporated into knowledge base

• Mark Miller-generated procedures incorporated into knowledge base

### Chosen Project Management Life Cycle

I have very good information about the project's goal and solutions. I also have low expectation of significant impact by outside factors. Given these two observations, I have chosen to use the Linear Project Management Life Cycle approach to this project.

#### Project Success Metrics

I believe that this project will be a success if the following metrics are met:

• Users will have documentation standards for both structure and composition. These standards will be understood by at least 90% of the student workers running the lab.

• Users--through single sign-on and cross-system categorization hyperlinks--will use 25% less time navigating the dual systems.

• Users will be able to search ticketing and knowledge base systems based on information categories.

• Knowledge base administrators will be able to moderate user submissions, and be alerted of the need for moderation.

### Issues to Resolve

I don't yet know how I'll implement the per-category and per-policy drill down features; once the user has chosen a category or policy, the knowledge base application can either call a whole new page containing the results, or use AJAX to display them on the current page. I suspect that the answer will come to me as I see how things are typically done in Ruby on Rails.

## Addendum

### Project Overview Statement

### Problem/Opportunity

*The Health & Informatics program maintains a computer lab (hence referred to as SIMMLAB) for HI students to become familiar with various software applications. Support contracts for the lab equipment are expensive, and will not be renewed. As such, it is important that the SIMMLAB become self-sustaining by renting its environment out to other schools teaching HI programs.*

*To that end, the system requires many changes to simplify management by students and add value to the environment. These changes constitute the overall "to-be" system.*

### Goal

*The project goal is to design and implement an information architecture that will support (and aid in supporting) the overall to-be system.*

### Objectives

*The project objectives are to:*

*• Standardize documentation by:*

*• Creation of a style guide for procedure write-ups, and*

*• Creation of templates for policies and their related procedures and task lists.*

*• Linking policies with their related procedures and task lists.*

*• Implement a knowledge base for storing the documentation*

*• Provide moderation for new knowledge base articles, and alert lab administrators when an article must be approved.*

*• Implement a searchable ticketing system that will serve as an event log. If the solution to a ticketed problem does not appear in the knowledge base, it must be written up (according to the style guide) and submitted to the knowledge base.*

*• Provide integration between knowledge base and ticketing system, providing:*

*• Straightforward navigation between information categories (e.g from ticketing system "Network" category to knowledge base's "Network" entries, and vice-versa).*

*• Unified user accounts between the ticketing and knowledge base systems, such that users will not have to juggle differing usernames and passwords.*

*• Incorporate content of current SIMMLAB knowledge base, as well as documentation produced by Mark Miller in the course of his project (which involves documenting the as-is system).*

### Success Criteria

*I have selected the following success criteria:*

*• Users will have documentation standards for both structure and composition. These standards will be understood by at least 90% of the student workers running the lab.*

*• Users--through single sign-on and cross-system categorization hyperlinks--will use 25% less time navigating the dual systems.*

*• Users will be able to search ticketing and knowledge base systems based on information categories.*

*• Knowledge base administrators will be able to moderate user submissions, and be alerted of the need for moderation.*

### Assumptions, Risks, Obstacles

*Due to financial restrictions, both the ticketing and knowledge base systems will be based on open source. Given this, I am assuming that having access to the source code will allow me to make the changes in each required for cross-system integration.*

*The software I intend to leverage for the knowledge base is written in Ruby, a language with which I have little experience. I am therefore making an assumption that my programming experience will enable me to make the necessary code changes. There is a bit of risk here, but I consider it to be quite small.*

*The greatest obstacle to project success is my never having completed (nor attempted) a project in such a structured manner as is required in this case.*

*Prepared by John Fox*

*November 25, 2013*

# Appendix B – Testing

Every software project must be tested. Even the most skilled analysts and designers can make errors in the phases prior to implementation, and even the most skilled programmers can introduce software bugs. Because I didn’t learn about testplan until I’d already completed the application, I did not generate a formal testplan for the project.

However, that is not to say that the application was not tested—or that bugs were not found. I took two approaches to testing, and they were somewhat complementary.

## Field Validation

ASP.NET provides for a type of form control called a “validator”. One or more validators can be assigned to each of a form’s input fields. Throughout the application, I assigned a “Required Field” validator to each form field. I then tested each such form thoroughly to ensure that if form submission were attempted with any field blank, a “this field is required” message would appear.

In my testing, I did the following for each form: if it had three input fields, then I attempted to submit the form with fields 1, 2, and 3 blank. When I had assured myself that these were covered, I then tested them in combination:

* Field 1 filled, fields 2 and 3 blank
* Fields 1 and 2 filled, field 3 blank
* Field 2 filled, fields 1 and 3 blank
* Etc., until I had tested all possible combinations

## Common Testing of Application During Development

I had no schedule for this testing, but once or twice a week, when I needed a change from adding further functionality to the application, I took 20 or 30 minutes to test things out; adding categories, making sure they showed up in the category listing when documenting an event, creating new procedures and making sure that an email alert was sent and that the new procedure appeared in the listing of those waiting for moderation, et cetera. This informal testing did indeed turn up bugs.

Because I have known for years that bugs are inevitable, I would not swear that Kable is now 100% bug free—though I do suspect that it is. However, I have scripted and practiced a fairly complete application walkthrough, and the application performs as expected.

# Appendix C – Kable User’s Manual

## About Kable

Kable provides a means by which

1. Problem events in the SIMMLAB may be documented and categorized
2. Procedures to fix problem events may be documented and linked to corresponding policy and category.
3. Procedures may be search by category or policy in order to turn up the solution to a previously occurring problem event.

The intent is to make it easier to manage the SIMLABB by providing a means by which a wheel must be invented only once, rather than each time a problem occurs.

## Interface Conventions and Application Menu Tour

In Kable, all creation, editing, and deletion features have the same interface, whether they pertain to policies, procedures, categories, events, application configuration, or procedure moderation. Figure C-1 illustrates the basic layout of these interfaces.

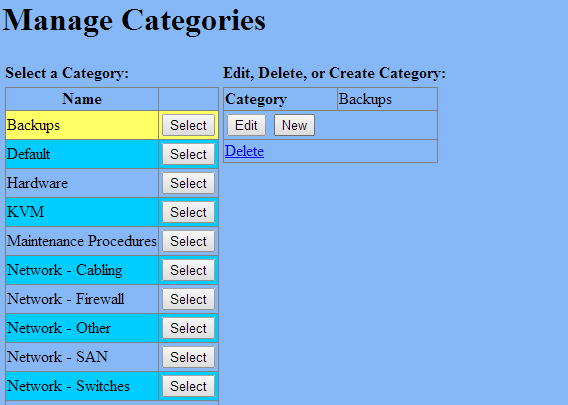


Figure C-:Example interface

In each interface, a gridview control on the left displays a listing of relevant items (event and procedure categories in this case). An explanatory caption above the gridview instructs the user to make a selection from the list, and a caption to its right explains what can be done once a selection has been made. When a selection is made, its data appears in a dataview control. At the bottom of the dataview control are buttons for operating on the selected item, or creating a new item. If the user wishes to delete the selected item, she clicks the “Delete” linkbutton and answers “yes” to the confirmation prompt that appears. If she wishes to edit the selected item, she clicks the “Edit” button, makes her changes, and confirms the edit by clicking the “Update” button that appears. If the user wishes to create a new item, she clicks the “New” button, enters the requested data, and confirms the new data by clicking the “Insert” button that appears.

A menu runs across the top of every Kable screen. It provides access to Kable’s administrative features, its events-and-procedures features, and to documents that aid in the formulation of policies and procedures. This menu provides the means to navigate the application. It is shown in Figure C-2.

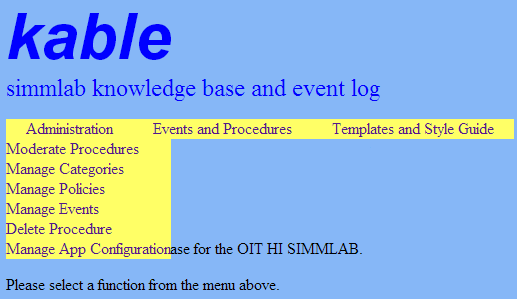


Figure C-2: Kable navigation menu

## Administrative Functions

Kable provides the following administrative features:

* Moderate Procedures
* Manage Categories
* Manage Policies
* Manage Events
* Delete Procedures
* Manage App Configuration

“Moderate Procedures” presents a list of unmoderated procedures. It may be sorted by category, summary, or creation date. To view an unmoderated procedure, click its “Select” button. If the procedure content is acceptable, click the “Approve” button, and confirm your approval by clicking the (what else?) “Confirm Approval” button.

“Manage Categories” allows the administrator to add, remove, or edit the categories that are associated with events and procedures. Any category associated with a procedure or event may not be deleted.

“Manage Policies” allows the administrator to edit, create, or delete a policy. To aid in the creation of policies, a policy template may be found under the “Templates and Style Guide” menu. The template body can be cut-and-pasted into the “Policy” field when creating a new policy. Note that the policy’s title should not appear in the “Policy” field, but in the “Title” field.

“Manage Events,” allows for the creation, editing, and deletion of events. This page is present mainly to provide a means by which events can be deleted (users cannot delete events) but because this page was based on the user-version of the Events page, there was no extra time cost involved in giving the admin version these creation and editing powers.

“Delete Procedures” provides a mean for the administrator to remove obsolete procedures.

“Manage App Configuration” is an interface for setting the applications two configuration parameters, both of which allow the system to send the administrator an email alert when a procedure has been created or edited, and is thus in need of moderation. These parameters are “adminEmail” (the email address that will receive the “moderation required” email) and “smtpServer” (the mail server that will be used to send the “moderation” required email. To modify one of these values, select it in the gridview, click the “Edit” button in the dataview, make your change, and click the “Update” button.

## User Functions (Events and Procedures)

Kable provides the following user functions for Events and Procedures:

* Create, Edit, or Browse Events
* Create or Edit Procedure
* Find Procedures by Category or Policy
* Browse Policies

**Create, Edit, or Browse Events**

In Kable parlance, an event is a problem that has occurred in the SIMMLAB environment. Perhaps a server backup failed, or user’s password has been forgotten. When such a problem is reported to you, you use this feature to document the event and associate it with an event category. Figure C-3 illustrates an event’s documentation.

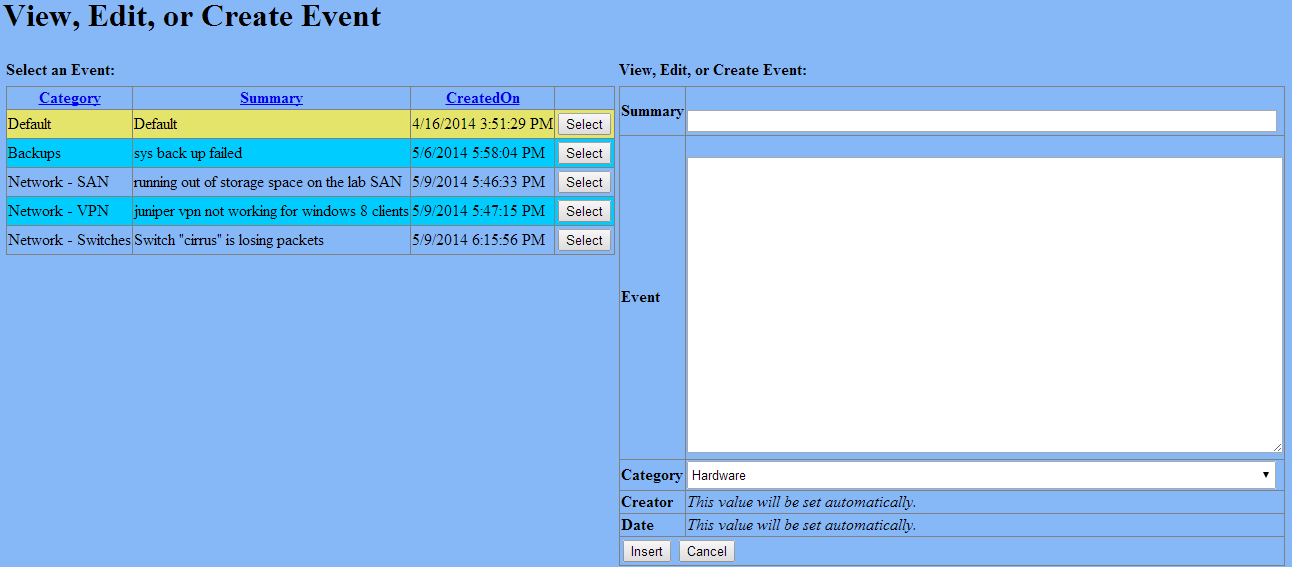


Figure C-3: Create new event

To create a new event, select an existing event from the list on the left. This will bring up a summary of the event details on the right. Scroll to the bottom of the dataview showing the event details, and click “New”. You will see the blank form shown in Figure C-3. Enter a brief summary in the “Summary” field, a full description of the event in the “Event” field, select an appropriate category from the “Category” dropdown list, and click “Insert”.

**Create or Edit Procedure**

Once you’ve identified how to resolve the problem documented in an event, use this function to make a record of the steps you took to solve the problem. When you load this page, you’ll see a listing of procedures on the left. To create a new one, select any in the list, and its details will appear to the right. At the bottom of the detail listing, click the “New” button.

Each procedure must be formatted according to a predefined template (see “Templates and Style Guide” below). Here is the procedure template:

*Procedure: <title, e.g. "Chronos Backup Configuration">*

*In keeping with policy "<title of relevant policy>," <general introduction to the topic>*

*Instructions: How to <task name, e.g. "Configure Chronos">*

*<instructions in either list or paragraph form>*

The placement of items within angle brackets (i.e. “<”, “>”) indicates material that should be customized for the procedure being documented. Figure C-4 shows a completed template filled out in the “New Procedure” form. Note that the “title” element does not appear in the list of instructions, but in the “Procedure” field at the top of the form.

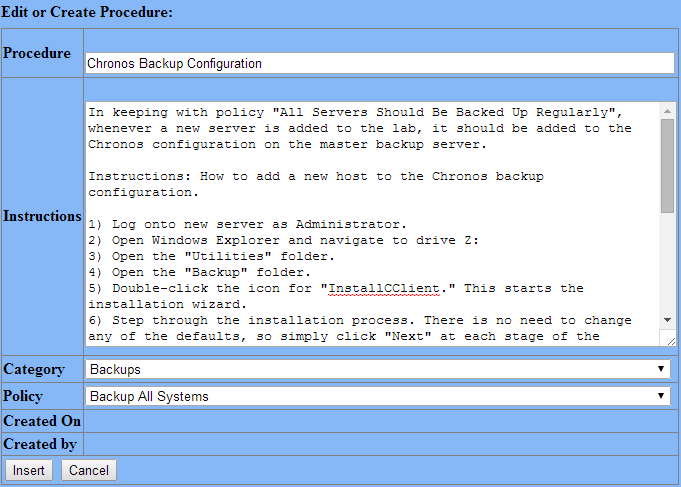


Figure C-4: Procedure filled out according to template.

There is one other thing to keep in mind when composing a procedure. A style guide has been provided to aid making procedure write-ups easier to both write and read. It is important that you read this guide, and make your best attempt at following its simple advice.

**Find Procedures by Category or Policy**

Sometimes, SIMMLAB events will recur. Perhaps, for example, the backup server has a bug that causes it to lose its configuration sometimes. When such an event recurs, its solution can be found by finding the appropriate procedures. When you navigate to this page, you will see a listing of procedures, as shown in figure C-4. Across the top of the procedure listings, are clickable hyperlinks labeled “Category”, “Policy”, “Summary”, and “Created On”. Clicking on any of these changes the order in which the procedures are listed, making it easy to find all procedures categorized as “Backups”. Similarly, you can sort on the associated policy or on the procedure’s summary, if that will help you find the procedure you’re looking for.

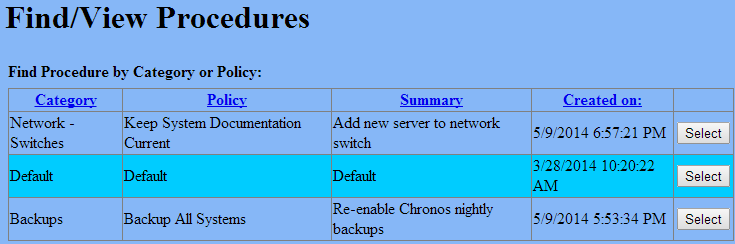


Figure C-5: Finding procedures

**Browse Policies**

At this point, you probably already know how to browse the list of policies. Load the page and select the policy you’re interested in reading. You may sort the list by title, creation date, or creator.

## Templates and Style Guide

The templates and style guides previously mentioned are available from the menu link “Templates and Style Guide”. The templates share a page, and the style guide has its own.

## Slainte!

I hope that one day a SIMMLAB administrator will read this manual. If you, kind reader, fit that bill, I hope you find this tool useful—or at the very least, it doesn’t drive you crazy.

Cheers!

1. In the spirit of intellectual honesty, I should point out that this consideration did not occur to me originally. It came about due to circumstances that I discuss in the “Project Implementation” portion of this document. [↑](#footnote-ref-1)
2. I chose “Rails” not because I’d ever done anything significant with it, but because I’ve read a fair amount about it over the years, and I thought it would be fun. Perhaps this was not the best criterion for choosing a development platform! [↑](#footnote-ref-2)
3. As stated earlier in this document, I’d intended to use Windows authentication to control access to the application. When it came time to deploy the application to IIS and enable Windows authentication, I found I’d hit a brick wall. I spent 20-30 hours trying to make it work, and I found plenty of information online that helped me get started, but no information that would let me finish. I was forced to wind the project up without determining the proper IIS settings for making the Windows authentication work. Currently, the application hard-codes dummy usernames in cases where the actual username would appear—had Windows authentication been completed. If someone takes on the tasks of making Windows authentication work, then those hardcoded “user” assignments will have to be updated. They can be found by searching the aspx.cs files for the word “TODO”. [↑](#footnote-ref-3)