University of Idaho
CS481: Senior Design

IDAHO Time, Accounting, and Reporting System

Department *of* Health / Welfare

prepared for

Don Moreaux
Marj Sanderson

and

The Idaho Department of Health and Welfare

Authors:
Scott Beddall
Brett Hitchcock
Chaylo Laurino
Alex Nilson
Zeke Long
Nathan Mazur

Advisors:
Greg Donohoe

May 10, 2012

Contents

1 Introduction

1.1 Identification

This Software Design Document pertains to the Idaho Department of Health and Welfare Time, Accounting, and Reporting System. Project development for Fall Semester, 2011 is executed by Scott Beddall, Brett Hitchcock, Chaylo Laurino, and Alex Nilson. The advisor for the project from the University of Idaho is Gregory Donohoe. The project sponsor and primary client from the Idaho Department of Health and Welfare is Don Moreaux.

1.2 Document Purpose, Scope, and Intended Audience

1.2.1 Document Purpose

This document's sole purpose is to outline the scope of Idaho TARS. This outline includes, but is not limited to:

- Development Decisions and Rationale.
- Architectural Specifications.
- Detailed Design Information.
- Locations of Other Project Resources.
- Complete Details Regarding Project Deliverables

1.2.2 Intended Audience for Document

TARS is to be mostly completed by the 5/9/2012. With that being the case, this document is aimed at any future developers or users of Idaho TARS.

1.3 Software Purpose, Scope, and Intended Users

1.3.1 Software Purpose

Idaho TARS is intended to provide time and resource tracking for contractor/non-contractor work efforts within the Idaho Department of Welfare. Work efforts must be added to time-bounded project PCA codes and approved by users with sufficient privileges. Project summaries, cost totals, user logs, and other information will then be available within TARS to authorized users. These users will be authenticated by an Active Directory interface.

1.3.2 Software Scope/Context

The Idaho Department of Health and Welfare is currently utilizing a resource called Mariner for time management and accounting. The IDHW's needs, however, are significantly less than the capabilities that Mariner provides. Portfolio and Resource Management, Planning, and other features of Mariner are being paid for, but left unused. The under-utilization of Mariner, coupled with fact that the IDHW would prefer an open-source solution that meets their specific needs and workflow, motivated Don Moreaux to bring the TARS project to the University of Idaho.

1.3.3 Intended Users for the Software

Intended Users of Idaho TARS are the staff and employees of the Idaho Department of Health and Welfare as well as its contractors.

1.4 Definitions, Acronyms, and Abbreviations

MVC	Model View Controller. A design pattern used for content focused websites. Provides security through modularity, ease of maintenence, and a clear architecture.
TARS	Time, Accounting, and Reporting System.
PCA Code	Position Classification Allocation Code.
SQL	Structured Query Language. Used for input and retrieval of data from a SQL database.
IDHW	Idaho Deparment of Health and Welfare
Work Effort	A project. Has one or more assigned PCA Codes and a list of associated work tasks.
Connection String	A formatted line of text that contains all relevant connection information for a remote resource. (server, dsa)
LDAP	Lightweight Directory Access Protocol. A protocol used for interface with distributed directory services. (Active Directory, Apache Directory Server)
DSA	Directory Service Agent. A server that specifically listens for queries via LDAP.
Active Directory	Microsoft Directory Services
Apache Directory Services	Open source LDAP alternative to Active Directory. Highly stripped down.
Task	An individual part of a Work-Effort
I-Time	Idaho Time - The system by which hours are actually submitted to the Idaho Government
Earnings Codes	Code describing a unit of work. VAC = Vacation. Etc.
IIS	Internet Information Services - Microsoft Server Infrastructure
.NET	Proprietary Microsoft Framework.

1.5 Document Overview

Section 2 describes software constraints imposed by the operation environment, system requirements, and user characteristics. After this it will identify the system stakeholders and list/describe their concerns.

Section 3 of this document describes the system and software architecture from several viewpoints, including, but not limited to, the developer's view and the user's view.

Section 4 provides detailed design descriptions for every component defined in the architectural view(s).

Section 5 provides traceability information connecting the original specifications (referenced above) to the architectural components and design entities identified in this document.

Section 6 is a complete overview of project deliverables as well as providing an overview of the Visual Studio Solution that makes up TARS.
Section 7 covers future work that will need to be done to finish the complete list of TARS requirements.
Sections 8 and beyond are appendices including code documentation and original information and communications used to create this document.

2 Software Requirements, Constraints, and User Characteristics

The following is a compiled list of TARS requirements. Below each individual requirement is a short comment as well as a status of that requirement as of 5/8/2012.

Current Idaho TARS Requirements

PCA -1
Users with the proper permissions must be able to manually enter PCA codes in a form that meets DHW standards.

Status: **Done**

PCA-2
Users with proper permissions must be able to manually tie work effort(s) to valid PCA.

This provides the ability to correctly connect the PCA code(s) to work, and ensures we are correctly accounting for what costs are allocated to what work. Work can be project related, a maintenance activity, or a non-productive activity such as meeting time.

Status: **Done**

PCA-3
The system must provide a mechanism for time bounding PCA codes - with the ability to "deactivate" a code prematurely and an open "end" date.

Codes need to have start and end dates assigned to them, and depending upon user permissions, those dates may or may not be editable. Do not mimic MS Projects start and end date functionality.

Status: **Done**

PCA-4
There is no PCA-4

PCA-5
Must maintain an audit trail (history of changes - people, projects, and PCAs).

Maintain history - history must remain static, dates, time (hours) cannot change once booked (accounting term for in system).

Status: **Done**

PCA-6
The system must provide a mechanism for preventing time to be allocated to expired PCA codes.

No Comments

Status: **Done**

PCA-7
The system must allow multiple PCA to be assigned to a work effort, over the life of the effort/project.

No Comments

Status: **Done**

PCA-8

Must be able to assign one or more PCA codes to work effort (split % allocation across multiple PCAs which can change during life of work effort).

In the case where a work effort is tied to more than one PCA, there needs to be some mechanism for determining how the work is partitioned between the two.

Status: **In Progress. Multiple PCA codes can be assigned, but % allocation is not available yet.**

PCA-9

Must allow work to be assigned to other entities outside DHW.

Provides a means of identifying between DHW and non-DHW contractor hours.

Status: **Done**

PCA-10

Must allow work to be associated with multiple divisions or the enterprise.

No Comments

Status: **Done**

DATA-1

The system shall track date specific vendor and employee/contractor information.

Provide some mechanism for tracking employment status and changes. For example, when a contractor is hired on as state staff, or changes vendors, TARS would allow that information to be entered and tracked. Need data elements.

Status: **Done**

DATA-2

The system shall allow for some description of work or project to be entered and attached.

Provide ability to describe the work effort in general terms.

Status: **Done**

DATA-3

The system shall be consistent with I-Time data.

Use the codes and logic from I-Time (see DAT-7 and REP-2) codes are Earning Codes - 3 digit

Status: **Done. When an employee adds a Work Effort to their timesheet to log hours to, they must also select an I-Time earnings code, which is saved as part of the hours entry.**

DATA-4

The system shall provide a means to replicate last week's assignments (repeating tasks can auto fill)

For those staff/contractors that repeat most work efforts each week, having the ability to replicate the proceeding week saves data entry time.

Status: **Done**

DATA-5

The system must have a method that allows staff to create work effort, and self-assign

Users would have the ability to create work effort, then assign themselves to that effort. Clarification - Staff can self assign to a particular work effort, Mgrs and Admin only can CREATE a work effort with a PCA

Status: **Done**

DATA-6

Must be able to track work effort for resources, depending upon their assignment, that are either cost allocated or not cost allocated.

Clarification - Cost allocated codes are PCA codes (all work is entered unto I-time as ACT, all non-work time captures is by Earnings Codes

Status: **In Progress.**

DATA-7

Must be able to break time out by time codes for NON-work efforts, such as Vacation, Sick, LWOP, (match I-Time data since this is the system of record)

Need to understand data requirement for I-Time if the plan is to eventually interface with I-Time. All time in I-time is seen as worked or non-worked. All "work" is coded to ACT, non work is coded to various codes.

Status: **Done? When an employee adds a Work Effort to their timesheet to log hours to, they must also select an I-Time earnings code that is saved as part of the hours entry.**

DATA-8

Users shall have the ability to close tasks and activities on their timesheet, and reopen if needed.

This is separate from the open/closed PCA codes. A user may no longer be working on a particular project or investment so they want to close it out on their time sheet.

Status: **Done**

DATA-9

The system shall provide some mechanism (configurable dropdown) for grouping of business, program, and function of work.

Need to be able to add/edit/delete values into those lists. This is grouping for a work effort

Status: **In Progress**

DATA-10

Audit trail data shall include the information that was updated, modified/deleted, date created, and by whom for each item determined to be auditable. The TARS team has proposed storing every SQL query executed by a user.

Need to define what will be included in the audit trail.

Status: **Done**

DATA-11

Data for staff and projects shall include the ability to store links and attachments

Related in part to DAT-2, in that it provides a means for capturing work and project information that describes the work effort.

Status: **No longer a requirement**

DATA-12

The system must allow for future time entry

Any user should be allowed to enter time against a work effort, in advance of the current week.

Status: **Done. No restriction is made on how far ahead time can be entered for, but there has to be a Work Effort to assign to.**

DATA-13

Must prevent work efforts to exist in the system unless they are tied to a PCA code.

PCA codes and work efforts (tasks, …) are all time bounded in this system. To prevent inaccurate recording of time allocated to an effort, some automated process of preventing expired or deactivated objects should be developed. System must also allow non-work time to be recorded using Earnings Codes (I-Time) such as VAC, HOL, etc.

Status: **Done, but still need a cleanup method for expired or deactivated objects**

REP-1

All data for reporting shall be extracted via external source (EDW. Excel, etc.).

Team sees no need to build in reporting in TARS, since we can generate reports with Business Objects, or other database connections.

Status: **Inactive. This was declared out of the scope of this semester and has not been considered at all yet.**

REP-2

Must allow users to create a view of their I-Time timesheet.

I-time is a separate timesheet, into which users also enter time for payroll accounting. (This requirement will be prioritized at the very bottom. Isn't needed until we interface with I-Time)

Status: **Inactive. This was declared out of the scope of this semester and has not been considered at all yet.**

REP-3

Reports must be real-time, reliable, and accurate. Includes exports to csv, Excel.

Speaks to having a simplified database schema, one that allows external connections (ODBC, etc.) to easily connect and extract data for reporting purposes. Reports will be created using Business Objects, not in TARS

Status: **Inactive. This was declared out of the scope of this semester and has not been considered at all yet.**

REP-4

The system must prevent time entry to a user's time sheet once the week period has cycled through. (a week is a calendar week, and cycled through means that the time sheet has been approved by management) Admin role should have the ability to "unlock" the time period and allow time sheet changes.

(Note: This is not to be confused with the open/close time periods of PCA codes)

Status: **Done**

VIEW-1
Must have a sort and group function that allows work effort to be grouped by application, division, manager, etc.

No Comments

Status: **In Progress.**

VIEW-2
The system must allow a user the ability to create a custom view of the data.

Users should be able to slice data, such as work effort by staff member over a date range. Users should only be able to see and customize their own data unless they are mgrs or admin and the view should persist.

Status: **Partly done. More custom views could be added though.**

VIEW-4
Must allow users to easily size windows

No Comments

Status: **Done. (Since it's a web application, browser windows can be resized)**

VIEW-5
Must be able to limit view of information presented to user to what is pertinant to that user's role.

This requirement is tied to VEW-2 in that it limits the range of customization of a view.

Status: **Done**

VIEW-6
The system shall provide search/find functionality to locate work efforts, with minimal amount of navigation (task actions<=4 clicks/pages/dialogs)

The number of clicks should be proportional to the frequency of the TARS tasks. In other words, TARS tasks that users frequently execute, should have the fewest navigation steps. Users can only create a view of their own data. Mgrs should be able to group their staff for time approval.

Status: **Done**

SEC-1
Must authenticate using Active Directory

Status: **In Progress.**

SEC-2
Must have a role-based permissions security.

Would like to have the ability to create new roles, and assign permissions to that role. For example, an Administrator have rights to edit/delete PCA codes and users, while an Individual Contributor would not have those rights. Basic role set would include Administrator, Manager /Approver, and Worker.

Status: **Done**

SEC-3
The system shall allow for automated closure of time periods for PCA and work efforts, with administrator ability to manually reopen & close for edit & approval

Tied to DAT-13 in preventing work efforts and codes from lingering when they are no longer active. This is also an example of a permission element in the role-based security profile.

Status: **Done. Work efforts can't be added to expired PCA codes, but Admin can edit the PCA codes to extend the date range**

NAV-1
The system must allow each user the ability to navigate easily by logic/functional areas, ie. Staff demographics, projects, work items/areas, time entry, etc.

No Comments

Status: **Done.**

NAV-2
Must automatically display current week when entering timesheet data.

No Comments

Status: **Done**

WKF-1
Must have notifications (via email, context, …) triggered by certain events such as timesheet submittal, approvals, PCA expiration, etc.

The current system auto-sends emails to notify users of their due timesheets, though the message is not tied to timesheet status (i.e. you get the mail even if your timesheet was submitted for that week). One notification would be to the worker who has NOT submittee his/her time by end of day on Saturday.

Status: **Coding for triggered events and seding emails is done and tested. Still need to change the sender email address (in Web.config) and write getEmailAddress(string userName) in UserController to get the recipient email address (and uncomment the call to it in SendEmail()).**

WKF-2
Users with permissions, must have the ability to approve TARS weekly submittals

Assumes ability to view other's timesheets based on your role permission. Rejected timesheets will provide notification to the submitter.

Status: **Done**

3 Software Architecture

3.1 Server Architecture - Microsoft Internet Information Services 6

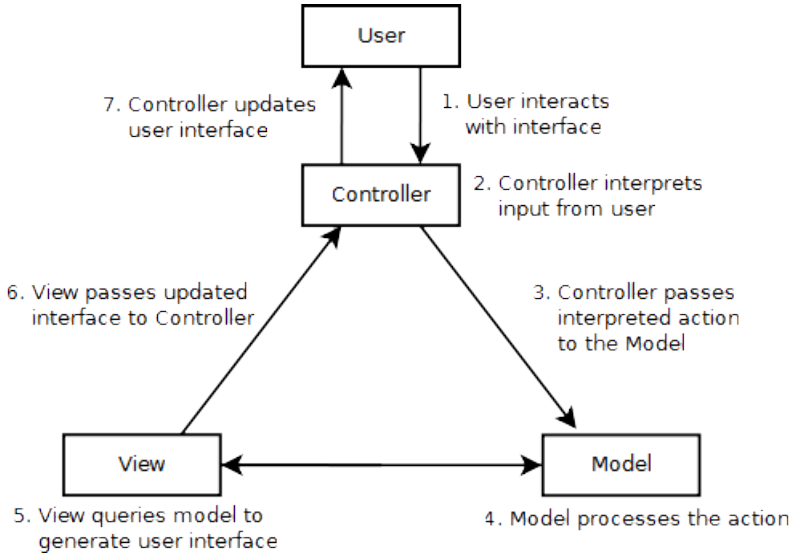
One of the IDHW's requirements for the this project is the use of Windows Server infrastructure. In this case the TARS development team will use Microsoft IIS6. Microsoft's Internet Information Services server is a modular, intuitive server application. New considerations must now be applied, however. IIS7, being a Microsoft product, uses Microsoft development software. Namely:

- C#
- ASP.NET
- Visual Basic/VB.NET
- .NET Development Framework
- Active Directory Services

TARS will be developed using all these technologies, as well as the IIS6 Model-View-Controller application. The MVC application will be described in detail in the next section.

3.2 Model-View-Controller

Most of the heavy-lifting for TARS will be present in the display and interaction with large amounts of data. This problem is what the Model-View-Controller design pattern was created for. The idea is that each word in the acronym: Model, View, and Controller each represent a component that handles a different aspect of the display process.



Advantage of Model-View-Controller architecture:

- Separates the user interface from the logic used in the database.
- Allows for independant development, testing, and maintenance of these seperate parts of the application.

3.2.1 Controller

A Controller receives input from the user and converts it to instructions for the Model and View components. Most MVC's use the first argument after the website URL as the controller call. In our case:

```
http://idahotars.com/home
```

Will load the "home" controller default function (Index). The second entry in the url after the controller selection will select the specific function in the controller. Any subsequent entries will then make up arguments to the specific function.
Load the viewTimeSheet function in the user controller:

```
idahotars.com/user/viewTimeSheet
```

Load the viewTimeSheet function in the user controller with arguments 10 and 20:

```
idahotars.com/user/viewTimeSheet/10/20
```

This configuration can be viewed or modified in "global.aspx."

3.2.2 Model

The Model manages database queries and assembles data for use elsewhere in the MVC. However, in the MVC3.0 extension specific to IIS6, there is a slight oddity when it comes to the Model component. That is, the Model doesn't actually execute any of the database interaction. Instead, the Models are used specifically as database table schemas, giving TARS a way to know what the remote tables look like. The actual interaction is carried out by Database Contexts. (DbContext class)

3.2.3 View

Views render the data received from the Models into viewable web pages.

3.3 SQL2008 Database

Though IIS can utilize any format of SQL Database, the IDHW requires that TARS use SQL2008. Any queries made by the MVC will be carried out by the "Model" component of the MVC.

3.4 Active Directory

The IDHW uses Microsoft Active Directory for their user authentification. With that infrastructure already in place, it is logical to use the same for Idaho TARS. To authenticate users of TARS, there will be three new Active Directory groups added: TARSAdmin, TARSManager, and TARSUser. If a given user is part of any of these three groups, they will be allowed access to TARS based upon their group.

To provide this functionality, the TARS development team wrote a helper class that can be used when an Active Directory connection is needed. This "LDAPConnection" class can be invoked within any controller, so long as "TARS.Helpers.LDAPConnection" is referenced.

Primary functions for now are:

```
LDAPConnection() //constructor, initiates variables needed for a connection
boolean requestUser(string user, string password); //returns true if the user/password combo exists
string requestRole(string user); //returns which group a user is part of
```

All these functions create an LDAP connection, query the DSA, and close the connection. The user of this helper class need not ever understand what is going on under the hood to use it correctly.

3.5 Security

While Idaho TARS will be used internally, there is still a security risk. The system may not be dealing with any highly confidential info, but TARS will have access to government resources like the IDHW Active Directory and I-Time interfaces.

To prevent easy exploitation of the TARS database, all queries to the database will be centralized in Model. This will not only make the team's code simpler, but also make it more secure. Centralized queries allow us to easily adopt a strong security stance. In addition, TARS uses DbContexts for database interaction. With that being the case, no SQL is ever directly executed from TARS. This significantly reduces the risk of SQL Injection attacks.

3.6 Browser Interface

Though it is probably already indicated by the server architecture, the development team must make it clear that this software is being developed for a web interface. This will eliminate many of the dependencies that are inherent to a system launched from a binary. The development team is developing this project to meet the following end system requirements:

- 1024x768 monitor resolution
- Google Chrome
- Mozilla Firefox
- Internet Explorer 7 and up
- Safari
- Compliant with W3C standards

4 Design Descriptions

4.1 Model-View-Controller Modules

4.1.1 Global Scripts and Config Files

There are two extremely important configuration files within the Visual Studio solution. Unfortunately, they are both named "Web.Config."

The first Web.Config is present in projectBase/Views/Web.Config. This config file handles various view configurations that affect TARS visually.

The second Web.Config is present in projectBase/Web.Config. It handles project level configuration; most importantly including the extremely important database connection strings.

By far the most important config file is projectBase/Global.aspx.
This contains the functions that control routing for the entire MVC. This job includes determining routing to controllers, filtering requests, and registering the initial routes on a default request for idahotars.com.

We also added three scripts files in the projectBase/Scripts folder:

1. DatePickerReady.js
    - Causes a calender to pop up for selecting a date when any editable DateTime variables with class=”datefield” are clicked on
2. Modal.js
    - Makes it easy to display any content (a string, table, image, etc.) in a stylized modal popup
    - Example: var modal = MyModalPopup(); modal.open({ content: ’enter content here’ });
3. Sticky.js
    - Makes it easy to display an unobtrusive notification
    - Example: $.sticky(”Timesheet Saved”);

4.1.2 Controllers

UserController, ManagerController, and AdminController are described in detail in Appendix C

- HomeController (only contains Index() and About() methods, which simply return a view)
- UserController
- ManagerController
- AdminController
- AccountController

The home controller is the default page in the case that a user is not logged in. It also provides the entry point to TARS when a controller is not specified.

AccountController provides login functionality. This includes calling the LDAPConnection helper class to authenticate users. AccountController was provided by a base example of an MVC, but is modified to serve the TARS specific needs.

UserController inherits from the default MVC Controller class as well as providing basic functionality for a normal user.

ManagerController inherits from UserController, but adds a couple more abilities that managers need as per the requirements specification.

AdminController inherits from ManagerController, inheriting all functionality as well as providing any and all administrative functions that are needed by TARS admin. Having these inherited privileges ensures that forced permission traversals will be almost impossible.

```
Home Controller //Default controller called when visited TARS
                //for the first time or when not logged in.

User Controller
 -> ManagerController
  -> AdminController
```

4.1.3 Models

- AccountModels (provided by IIS6 MVC)
- Divisions
- EarningsCodes
- Extensions
- History
- Holidays
- Hours
- PcaCode
- PCA_WE
- TARSUser
- Timesheet
- TimesheetRow
- WorkEffort

These items do not inherit from the default Model class. In addition, each class has an associated DbContext class in the same file. Example:

```
public class PCA_CodeDBContext : DbContext
{
   public DbSet<PcaCode> PCA_CodeList { get; set; }
}
```

These connections are not conventional classes or objects. To indicate this, their naming conventions are slightly more verbose. A DbContext creates a new context instance that utilizes an existing Database Connection String (located inside web.config) to create a connection to a SQL Database.

4.1.4 Views

- Every controller has an associated View. Located in their respective folders. Manager example:

```
Views/Manager/___.cshtml
   ///where ___ is a view associated with a given function within the controller
```

4.2 SQL2008 Database Schema and Interface Description

As mentioned above in the Software Architecture section, TARS will use a SQL2008 database to store all interactions other than User Info (handled by Active Directory). Before outlining the Database Schema, it would be wise to fully describe the thought process of the development team.

Stripped down to its most base parts, TARS is simply a database interface through which users can log and retrieve hours to and from work efforts. These "Work Efforts" are simply general projects that can have hours of contractor/non-contractor work added to their totals. For instance, there might be a Work Effort that is assigned its own unique ID and who"s description is "Document the latest changes to the TARS SDD." Any employees who wish to log their hours will find the Work Effort's ID, add their hours along with

other relevant data, and submit their entire timesheet for approval.

The Work Effort to "Document the latest changes to the TARS SDD." now has hours logged on it and waiting for approval. A user with the correct Active Directory permissions (part of group TARSManager or TARSAdmin) can now go check the status of the Work Effort, and approve any pending hours waiting on it. The development team has chosen to add all hours, approved or not, to the Work Effort's database table. A simple boolean present as a column in the table will ensure that filtering by approved/un-approved will be a simple task.

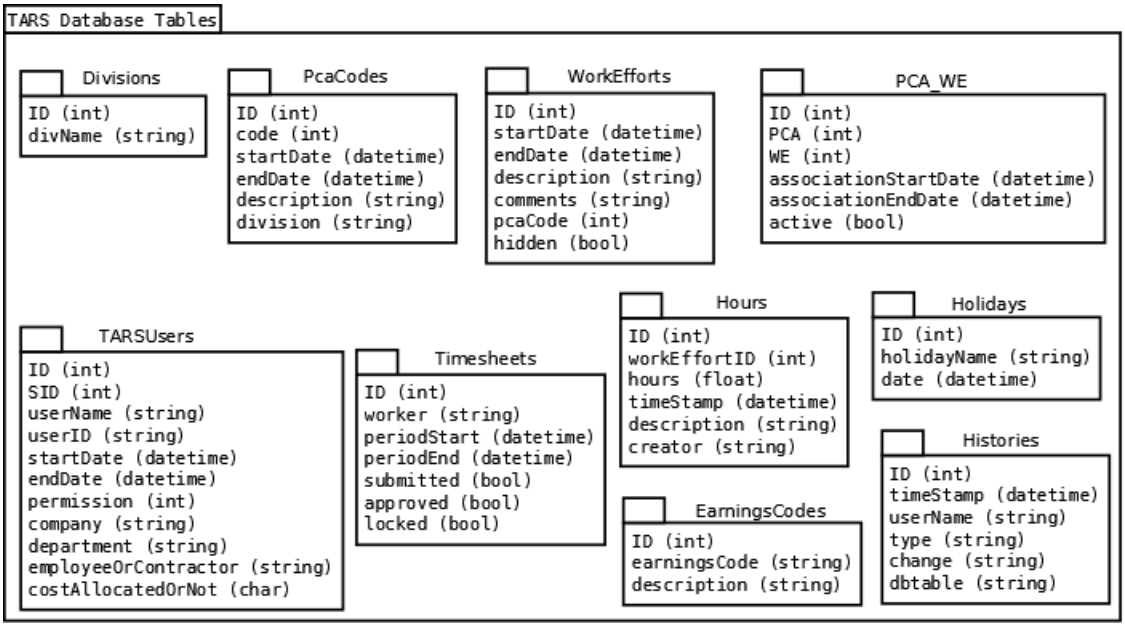
Unfortunately, the process is not done. Now that a Work Effort has hours charged to it, how can the accounting department charge these expenses? The simple answer is, they can't yet. To provide that functionality, PCA Codes must be assigned. This introduces another database table, as one PCA Code may have multiple Work Effort associations; just as one Work Effort may be associated with multiple PCA Codes.

One final note about the Work Efforts and PCA Codes is needed. The requirements state that they both must be time-bounded, capable of early expiration, and renewable.

In addition, for each of these Models, there is also a modification to their paramaterized DbContexts. That is, DB.saveChanges() is overwritten to save a new entry in the History table before calling super.saveChanges().

Addendum as of 11/23/2011. The TARS development team has learned that while the Active Directory DSA will service user/password requests, TARS will be unable to push any changes to the Directory. With that being the case, the development team quickly deduced that yet another database table would be needed. Namely, the "TARSUser" table. This table will store unique userids as well as any web and role configuration information that may be needed by TARS.

4.2.1 TARS Database Schema



4.3 Naming Conventions

All defined classes and objects will use capitalized camelcase, with their first letter a capital as well.

Local Variables and instances of classes will use camelcase also, neglecting the capitalization of their first letter.

```
class Manager : Controller
{
    public int id;
    public bool approved = FALSE;
    public string hoursType;
}
```

5 Traceability Information

All effort on the project may be tracked via the project GitHub repository: github.com/ICBM/TARS

The project website resides at: http://seniordesign.engr.uidaho.edu/2011-2012/CostManagement/

Client requirements changes will be handled via email. Current progress on requirements as well as the requirements changelog is available at: github.com/ICBM/TARS/blob/master/doc/requirements_summary.docx

6 Deliverables

6.1 Deliverables Overview

At the end of the first semester the TARS team turned over the following deliverables to be passed on to the next semester's team for completion; Most of this Design Document, Client Requirement Documents, Meeting Minutes, Tutorials, Prototype Source Files, Requirements Summary, and our GitHub Repository. For exact locations, check the "GitHub Locations of Importantance Section"

This Design Document contains all of the conceptual and technical designs for this project. It also includes various Client restrictions on software and platform for the project. For further details please read the document.

The Client Requirement Documents outline the Client's desired functionality of the system. It is broken down into six parts: PCA, Data, Reporting, View, Security, Navigation, and Workflow. This document was provided to us by the Client. It has been updated several times as requirements have been added or altered.

All of the Minutes of our meetings with our clients as also been added. They show the changes in the requirements over time and contain clarifications to questions the client may have had during the course of the project.

For the first semester team, extensive trial and error was involved in setting up our machines to be viable development platforms based on the restrictions laid out by the Client. To prevent the next team from having to deal with the same problems already dealt with, the TARS development team has created several tutorials to help smooth the process.

A functioning application has been produced. It includes all of the models, views, and controllers. The

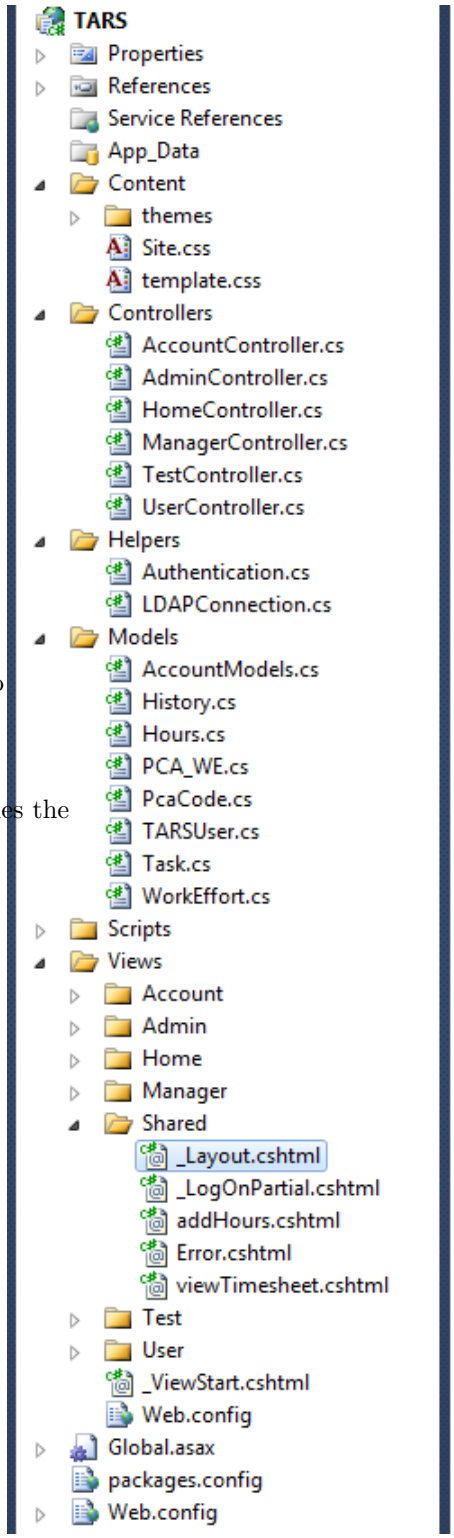
source code has been provided as a Microsoft Visual Studio solution. The full details concerning the functional state of TARS in relation to the full project requirements are laid out in the Requirements Summary document as well as the Requirements section of this document.

The Requirements Summary is a combination of the Client Requirement Document and a full summary of the application's functionality. It explains in detail the current status of every requirement.

The final deliverable is the TARS GitHub Repository. It has been the primary location for all collaborative work. It is an open source repository as the client desired to make the source code open-source. Maintaining private collaborators costs twelve dollars a month.

6.2 Prototype Deliverable Detail: Visual Studio Solution Architecture

At the topmost level of TARS are a variety of folders as well as configuration information for the project as a whole.



Web.config contains all configuration information for the webpage. The primary use of it so far has been to specify the "sender" address for notification emails, and to add Connection Strings to allow for database connection.

Content folder contains the CSS data for the website.

Controllers folder contains all Controllers for the project, which are described in Appendix C.

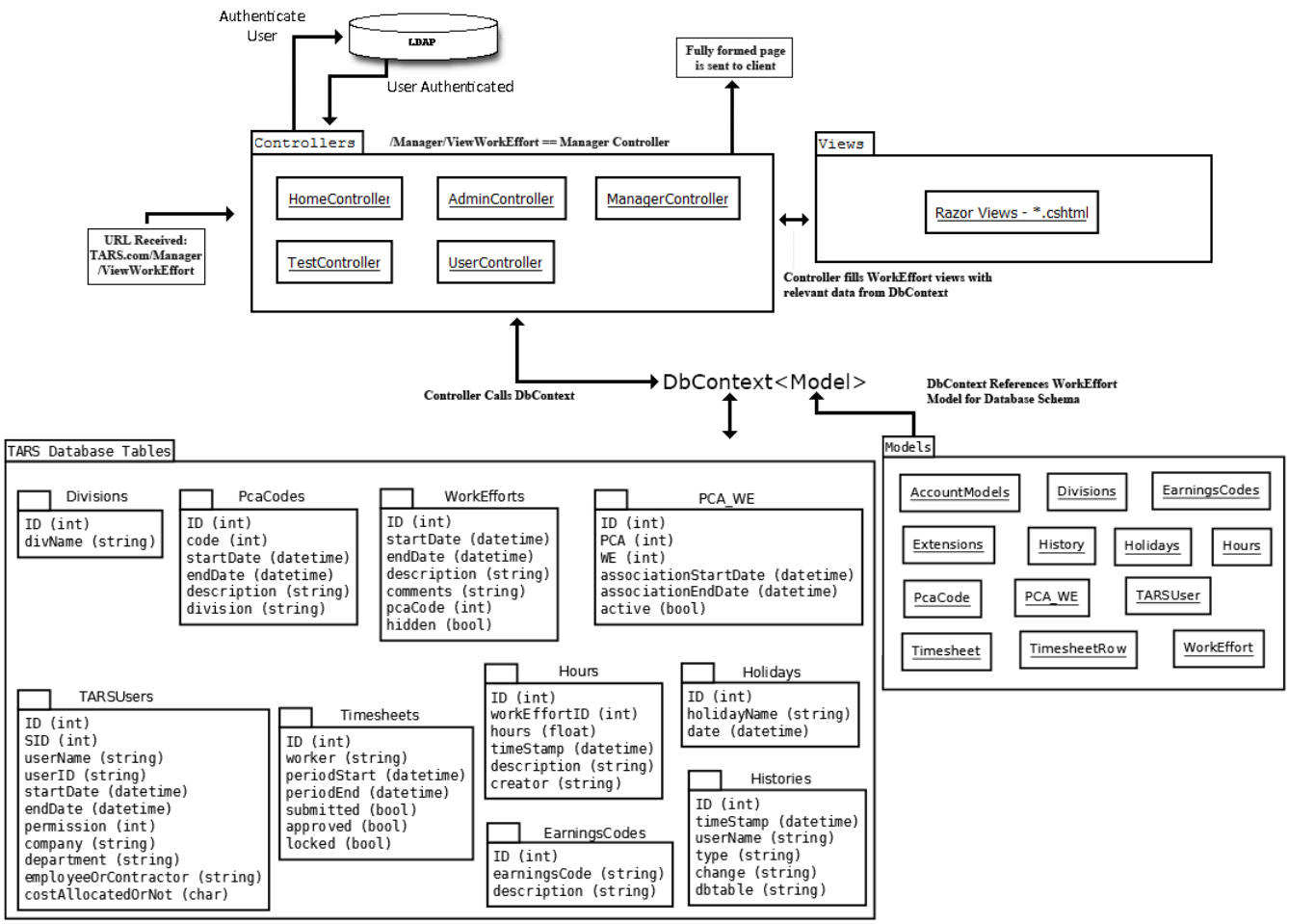
Helpers folder contains all helper classes under the namespace TARS.Helpers. Included thus far are the Authentication and LDAP-Connection classes.

Models folder contains all model classes, which are described above.

Views folder contains all relevant views, which are organized into subfolders according to the Controller they belong to. The exception to this is the Shared folder, where a controller will look for a view if it is not found in its own specified folder. This is very useful for views that are spread unchanged between permission classes, such as _viewTimesheetPartial. As well, it should be noted that _Layout provides the default layout for ALL pages.

So, for the server to generate a sample page to the user (let's say webroot/Manager/viewWorkEffort?id=1), it will follow this general process:

- Open the appropriate controller, in this case Manager. The server will only call the constructor when it is first created, not on every new page load.
- Check the Manager controller for a public function called viewWorkEffort. If this function were not found, it would check under the User controller, since Manager inherits from User.
- In this case, Manager has viewWorkEffort overridden from User to enable Edits, so it will load this function and check for proper arguments.
- The function declaration is expecting an *int id*, which we have passed. So now, the function will load.
- In the function body, the function will make an Authentication class and check to make sure the user logged in actually is a Manager or Admin. If not, it will redirect the User to an error or login page.
- To obtain information about Work Effort # 1, the function will create an empty WorkEffort class and fill it from WorkEffortDBContext.WorkEffortList.Find(1); Behind the scenes, WorkEffortDBContext is connecting to a server according to the connection string in Web.config, and looking for a table called dbo.WorkEfforts.
- At this point, any other information can be passed to the View by adding it to ViewBag. EX: ViewBag.newitem = "Hello."; //Runtime type checking on this.
- The View now has all needed information from the Model and ViewBag, so output everything in the View and display to the user.


Authenticate User
LDAP
User Authenticated
Fully formed page is sent to client
Controllers
/Manager/ViewWorkEffort == Manager Controller
HomeController
AdminController
ManagerController
TestController
UserController
Views
Razor Views - *.cshtml
URL Received: TARS.com/Manager /ViewWorkEffort
Controller fills WorkEffort views with relevant data from DbContext
Controller Calls DbContext
DbContext<Model>
DbContext References WorkEffort Model for Database Schema
TARS Database Tables
Models
AccountModels
Divisions
EarningsCodes
Extensions
History
Holidays
Hours
PcaCode
PCA_WE
TARSUser
Timesheet
TimesheetRow
WorkEffort
Divisions
ID (int)
divName (string)
PcaCodes
ID (int)
code (int)
startDate (datetime)
endDate (datetime)
description (string)
division (string)
WorkEfforts
ID (int)
startDate (datetime)
endDate (datetime)
description (string)
comments (string)
pcaCode (int)
hidden (bool)
PCA_WE
ID (int)
PCA (int)
WE (int)
associationStartDate (datetime)
associationEndDate (datetime)
active (bool)
TARSUsers
ID (int)
SID (int)
userName (string)
userID (string)
startDate (datetime)
endDate (datetime)
permission (int)
company (string)
department (string)
employeeOrContractor (string)
costAllocatedOrNot (char)
Timesheets
ID (int)
worker (string)
periodStart (datetime)
periodEnd (datetime)
submitted (bool)
approved (bool)
locked (bool)
Hours
ID (int)
workEffortID (int)
hours (float)
timeStamp (datetime)
description (string)
creator (string)
Holidays
ID (int)
holidayName (string)
date (datetime)
EarningsCodes
ID (int)
earningsCode (string)
description (string)
Histories
ID (int)
timeStamp (datetime)
userName (string)
type (string)
change (string)
dbtable (string)

6.3 GitHub Locations of Importantance

The root directory is present at github.com/ICBM/TARS

- SSD: /doc/ssd.tex – /doc/ssd.pdf
- Meeting Minutes: /minutes
- Tutorials: /doc/Tutorials-Resources/
- Prototype: /TARS
- Requirements Summary: /doc/requirements.docx
- Client Requirements Document: /doc
- Contact Information: /contact_info.pdf
- SQL Config Dumps: /doc/Tutorials-Resouces/SQL/
- LDAP Config Dumps: /doc/Tutorials-Resouces/LDAP/

7 Future Work

- Sending Emails
  - In Web.config, "smtp from", "host", and "userName" need to be changed to local server for system.net mailSettings
  - In UserController, getEmailAddress(string username) needs to be implemented, and the call to it needs to be uncommented in SendEmail()
- Sorting tables by clicking on column headings
- Allowing splitting percentage allocation for PCA Codes when a Work Effort has multiple PCA Code associations
- Functions that rely on Active Directory
  - Logins
  - Creating a user (an Admin function)
  - Editing a user's info, including end date, company name, etc. (an Admin function)
- Periodically cleaning database tables

8 Appendix A: Use Cases

Login

1. Click the "Login" button
2. Enter username and password
3. Hit Enter or click "Submit"
4. System authenticates and redirects to home page.

Adding new PCA code.

1. Select "Add PCA"
2. System loads PCA form
3. Fill in form, including time bounds for the PCA code
4. Press "Submit"
5. System updates tables and redirects to new PCA display page
6. A request to the financial department will be automatically dispatched. PCA codes are not actually generated by TARS.

Deactivating PCA code

1. Select desired PCA code
2. System loads PCA display page
3. Click deactivate
4. Confirm deactivation.
5. System updates tables and locks PCA code.

Adding Work Effort

1. Select "Add Work Effort"
2. System loads Work Effort form
3. Fill in form
4. Associate desired PCA code or codes, if multiple PCA codes are chosen a percentage of work effort may be set for each
5. Associate entity or entities
6. Click "Submit"
7. System updates tables and redirects

Updating Work Effort

1. Select "Update Work Effort"
2. System loads Work Effort form
3. Make Changes in form
4. Click "Submit"
5. System updates tables and redirects

Adding Hours

1. Select "Add Hours"
2. System loads Work Effort Selection form
3. Select Work Effort
4. System loads Hours form
5. Fill in form or select "replicate" to fill with previous weeks data
6. Click "Submit"
7. System updates tables and redirects

Approving Hours

1. Select "Approve Hours"
2. System loads hour approval form and fills with items needing approval
3. Select item
4. Select "Approve" or "Disapprove"
5. System updates tables and redirects back to hour approval form

Adding Tasks

1. Select "Add Tasks"
2. System loads Work Effort Selection form
3. Select Work Effort
4. System loads Task List form
5. Add desired tasks
6. Click "Submit"
7. System updates tables and redirects

Edit/Update Employee/Contractor Data

1. Select desired entity
2. System pulls data from database and loads Entity Form
3. Edit/Update as desired
4. Click "Update"
5. System updates table and redirects.

View History

1. Select "View History"
2. System loads History Search form
3. Enter Search Criteria
4. System loads Search Result form and fill with data

The system must allow for future time entry

1. Select "Add Work Effort"
2. Fill in form, making sure to add the effort to the correct date.
3. Click "Submit"

All data for reporting shall be extracted via external source (EDW. Excel, etc.).

1. Under a given PCA code or work effort, select "Get Data Report".
2. The system will then generate a copy of the data in EDW or Excel format.
3. User saves the copy at a destination of their choosing.

Must allow users to create a view of their I-Time timesheet.

1. User logs in.
2. On the user's personal page, select "Get I-Time Report".
3. The system will then generate a copy of the data for viewing.

Must have a sort and group function that allows work effort to be grouped by application, division, manager, etc.

1. User logs in.
2. User selects "My Work Efforts"
3. System generates list of all work efforts related.
4. User selects to sort by name, application, etc.
5. System sorts and redisplays work efforts in the proper order.

The system must allow a user the ability to create a custom view of the data.

1. User selects a PCA code or Work Effort code
2. User selects "Get Data Report".
3. User enters custom settings and hits "Select".
4. System generates report.

Must allow users to easily size windows

1. User resizes browser, which will resize the web interface.

The system shall provide search/find functionality to locate work efforts, with minimal amount of navigation (less than 5 clicks per important action)

1. User logs in.
2. User navigates to a work effort by...
    - Clicking on a link in the "My Recent Work" bar.
    - Entering a Work Effort code or PCA code in the Search bar on the top.

9 Appendix B: Database Details

9.1 Tables

ICBMDB Database:

- dbo.Divisions

  In TARS application terms, Divisions represent the highest level of organization. They reference no other data objects, but are used as a reference themselves. All divisions must be manually entered into the database prior to launch; there is no functionality in TARS to add divisions.

  Fields:
  - varchar(50) **divName**: This field is used to differentiate between different division objects.

- dbo.EarningsCodes

  In TARS application terms, EarningsCodes are three letter codes that are used to categorize an Hours object. An EarningsCodes object is used for display and categorization purposes, and is thus used only as a reference object. All EarningsCodes must be manually entered into the database prior to launch; there is no functionality in TARS to add EarningsCodes

  Fields:
  - nchar(3) **earningsCode**: A three letter code designating an earning code category
  - varchar(75) **description**: A short explanation of what the associated 3 letter code means in real world terms

- dbo.Histories

  The TARS application will keep a record of all actions that change the database in any way. This is an automatic function that runs in the background. The data in this table can be viewed by administrators, but otherwise cannot be interacted with in any way.

  Fields:
  - datetime **timestamp**: The date and time that the registered event occurred
  - varchar(50) **username**: The username corresponding the to the TARSUser object that initiated the event
  - varchar(50) **type**: The type of event being logged; add, delete, and modify are the three possibilities
  - varchar(max) **change**: List of object fields that were changed along with their new associated values
  - varchar(50) **dbtable**: The database table that was modified by the event.

- dbo.Holidays

  A Holiday object exists to allow the TARS application to account for three-day weekends during its timesheet-locking processes. Holiday objects must be entered by an administrator. They are only referenced by TARS timesheet locking functionality, and thus do not interact with other data objects.

  Fields:
  - varchar(50) **holidayName**: The name of the holiday in question
  - datetime2(7) **date**: The date of the holiday. The datetime2(7) type is more precise than the normal datetime type.

- dbo.Hours

  In TARS application terms, an Hours object is the primary object of interest. Hours objects are looked up in the database by means of the string value representing the username of their associated TARSUser object, the int value representing the ID of their associated WorkEffort object, and the datetime value representing the day the Hours were logged to. They are thus linked to the TARSUser object and the WorkEffort object.

  Fields:
  - int **workEffortID**: All hours must be attributed to a specific WorkEffort object. This field registers the ID of the associated WorkEffort.
  - float **hours**: This field represents the number of hours to be charged to the WorkEffort identified.
  - datetime **timestamp**: This field logs the date that the hours were worked for reference purposes.
  - varchar(65) **description**: This field tracks the 3-letter EarningsCode to categorize the hours logged, as well as the EarningsCode description associated with that code.
  - nchar(50) **creator**: This field will be set equal to the username of the TARSUser that created the object. In real world terms, this is the employee that the time must be attributed to.

- dbo.PCA_WE

  The PCA_WE object does not represent an actual entity; instead, it was put in place to allow multiple associations between PcaCode and WorkEffort objects. When a new association between a PcaCode and WorkEffort object is needed, a new PCA_WE object is created and saved to the database with information about the two objects that are newly associated.

  Fields:
  - int **PCA**: This field will have the value of the PCA Code that is part of the PcaCode object. It will be used to find the associated PcaCode object when necessary for display or record purposes.
  - int **WE**: This field will have the value of the
  - datetime2(7) **associationStartDate**: This field is present for record keeping purposes. It logs the date where the WorkEffort was attached to the PcaCode
  - datetime2(7) **associationEndDate**: This field is present for record keeping purposes. It logs the date where the WorkEffort and PcaCode were dis-associated (via deletion of WorkEffort by a manager or the association being removed by an admin).
  - bit **active**: This field is present to indicate the active/inactive status of the association between the WorkEffort and PcaCode objects.

- dbo.PcaCodes

  The PcaCode object contains a reference to the division it belongs to. One may thus always find a PcaCode object's associated Division object, and may also find all PcaCodes associated with a division by a database search. WorkEffort objects associated with a PcaCode object may only be found by searching the PCA_WE database table for all PCA_WE objects that reference the PcaCode.

Fields:

- int **code**: A five digit code representing a real-world PCA code. These values must be unique.
- datetime2(7) **startDate**: The date of the PCA code's creation. This may be set retroactively, and does not have to be done the day of the PCA code's creation in the real world.
- datetime2(7) **endDate**: The date of the PCA code's end. Hours may no longer be logged to associated WorkEffort objects (accessed via the PCA_WE association objects).
- varchar(40) **description**: A textual description of the real-world PCA, entered by the creator.
- varchar(40) **division**: This field will bear a value identical to the divName field of an associated Division object. This field creates and preserves the association, and can only be set to a value corresponding to an actual Division object.

- dbo.TARSUser

  The TARSUser object represents employees who have user accounts on the TARS system. Necessary data will be stored to preserve associations with other objects and to define their interactions with the TARS system. TARSUser objects are associated with both Hours and Timesheet objects.

  Fields:

  - varchar(184) **SID**:
  - varchar(40) **userName**: The username that the employee in question uses to log onto the TARS system. This field is also used to maintain associations with Hours and Timesheet objects. Hours and Timesheets associated with this user are found using this value.
  - varchar(26) **userID**:
  - datetime2(7) **startDate**: The starting date of a user's employment.
  - datetime2(7) **endDate**: The ending date of a user's employment.
  - int **permission**: An integer value corresponding to a user's permission level in the TARS system. 1- admin, 2- manager, 3- user.
  - varchar(40) **company**: The company for which the user works. Maintained for record purposes.
  - varchar(40) **department**: The department with which the user works. Maintained for record purposes.
  - char(2) **employeeOrContractor**: A value to indicate whether the user is an employee of IDHW or a contractor.
  - char(1) **costAllocatedOrNot**: A value to indicate whether the user has had their timesheets approved. It is referenced by functions associated with TARS automatic notification system.

- dbo.Timesheets

  Timesheet objects serve as a method of organizing Hours objects. Timesheet objects are generated for each user in 1-week segments. As timesheet objects are used to display hours, hours are therefore displayed one week at a time. Timesheet objects are associated with TARSUser objects via username. The timesheet is populated with hours

by going through the associated user and finding all Hours objects that have the same username.

Fields:

- varchar(50) **worker**: This field contains the value of the associated TARSUser's username. It is used to lookup Hours objects and populate the timesheet views.
- datetime **periodStart**: This field contains the beginning of the week that the timesheet corresponds to; it is used to filter Hours objects so that only those part of the corresponding week are represented in the views.
- datetime **periodEnd**: This field contains the end of the week that the timesheet corresponds to; it is used to filter Hours objects so that only those part of the corresponding week are represented in the views.
- bit **approved**: A flag to indicate whether the timesheet has been approved by a manager.
- bit **locked**: A flag to indicate whether the timesheet has been locked. Timesheets are locked automatically two days after a pay period ends. They may be unlocked by administrators.
- bit **submitted**: A flag to indicate whether the timesheet has been submitted for approval. This is changed when users submit their timesheet. It can also be changed to unsubmitted status by a manager who rejects a timesheet.

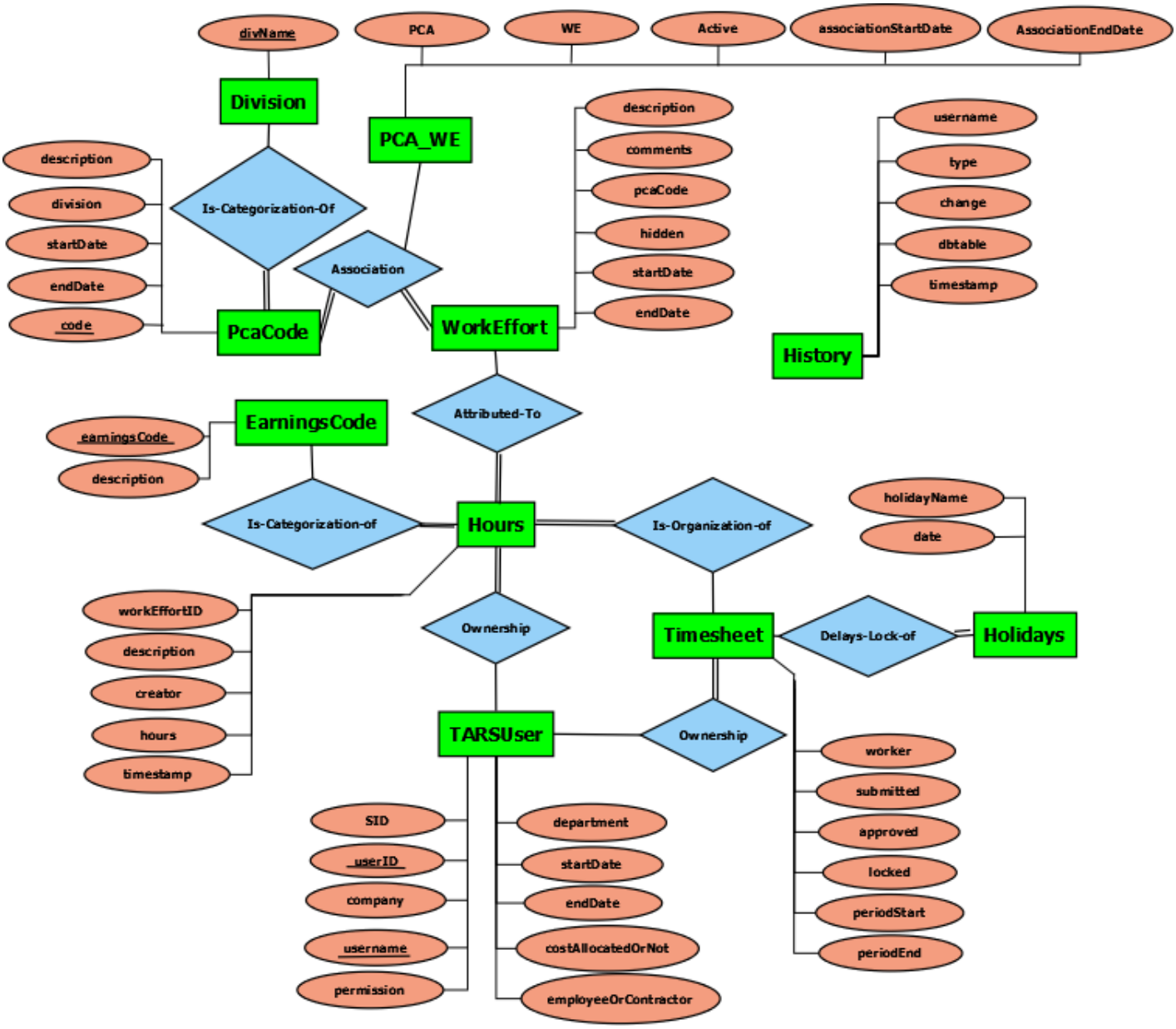
- dbo.WorkEfforts

  WorkEffort objects are linked to Hours objects as well as PcaCode objects. The former by means of a field in the Hours object and the latter by means of the PCA_WE model.

  Fields:

  - datetime2(7) **startDate**: The starting date of the Work Effort.
  - datetime2(7) **endDate**: The ending date of the Work Effort
  - varchar(100) **description**: A brief description of the real-world work effort.
  - varchar(250) **comments**: A field for the creator of the work effort to log comments.
  - Int **pcaCode**: The integer value of the initial PCA code that the work effort is attached to. Further PCA code associations will necessitate a PCA_WE object. However, every Work Effort must be associated with at least one PCA code.
  - bit **hidden**: A flag to indicate whether the work effort should be hidden or not.

9.2 Table Associations



10 Appendix C: Code Documentation

10.1 User Controller

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Net.Mail;

using TARS.Models;
using TARS.Helpers;

namespace TARS.Controllers
{
    public class UserController : Controller
    {
        protected WorkEffortDBContext WorkEffortDB = new WorkEffortDBContext();
        protected HoursDBContext HoursDB = new HoursDBContext();
        protected TimesheetDBContext TimesheetDB = new TimesheetDBContext();
        protected TARSUserDBContext TARSUserDB = new TARSUserDBContext();
        protected DivisionsDBContext DivisionsDB = new DivisionsDBContext();
        protected EarningsCodesDBContext EarningsCodesDB = new EarningsCodesDBContext();
        protected PcaCodeDBContext PcaCodeDB = new PcaCodeDBContext();
        protected PCA_WEDBContext PCA_WEDB = new PCA_WEDBContext();
        protected HolidaysDBContext HolidaysDB = new HolidaysDBContext();

        //
        //Index view, redirects to viewTimesheet function
        [HttpGet]
        public virtual ActionResult Index()
        {
            Authentication auth = new Authentication();
            if (auth.isUser(this) || Authentication.DEBUG_bypassAuth )
            {
                return RedirectToAction("viewTimesheet", new { tsDate = DateTime.Now });
            }
            else
            {
                return View("notLoggedOn");
            }
        }


        //
        /* Adds the newhours object to the database. Before saving, it checks to see if the hours' timestamp
         * is within the time frame of the work effort that the hours are being logged to.
         */
        [HttpPost]
        public virtual bool addHours(Hours newhours)
        {
            Authentication auth = new Authentication();
            if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
            {
                WorkEffort tmpWe = WorkEffortDB.WorkEffortList.Find(newhours.workEffortID);

                //make sure that the new hours are within the work effort's time bounds
                if ((newhours.timestamp < tmpWe.startDate) || (newhours.timestamp > tmpWe.endDate))
                {
                    return false;
                }
                //make sure that a timesheet exists for the period hours are being added to
                checkForTimesheet(newhours.creator, newhours.timestamp);
                //add and save new hours
                HoursDB.HoursList.Add(newhours);
                HoursDB.SaveChanges();

                return true;
            }
            else
            {
                return false;
            }
        }


        //
        // Returns TRUE if the date is within the start and end dates of the the Work Effort
        public bool isWithinWeTimeBounds(int weID, DateTime hrsDate)
        {
            WorkEffort tmpWe = WorkEffortDB.WorkEffortList.Find(weID);
            if ((hrsDate >= tmpWe.startDate) && (hrsDate <= tmpWe.endDate))
            {
                return (true);
```

```
    }
    return false;
}


//
/* Checks if a timesheet exists for the specified user and date.
 * If not, then createTimesheet() is called
 */
[HttpGet]
public void checkForTimesheet(string userName, DateTime tsDate)
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        Timesheet resulttimesheet = new Timesheet();
        DateTime startDay = tsDate.StartOfWeek(DayOfWeek.Sunday);

        //Check if there is a timesheet for the week that corresponds to newhours.timestamp
        var searchTs = from m in TimesheetDB.TimesheetList
                       where (m.worker.CompareTo(userName) == 0)
                       where m.periodStart <= tsDate
                       where m.periodEnd >= tsDate
                       select m;
        foreach (var item in searchTs)
        {
            resulttimesheet = item;
        }

        //if there isn't a timesheet for the pay period, then create one
        //If there is a timesheet for the current pay period, don't do anything
        if (resulttimesheet.periodStart.CompareTo(startDay) != 0)
        {
            createTimesheet(userName, startDay);
            return;
        }
        return;
    }
    else
    {
        return;
    }
}


//
//Creates an empty timesheet for the specified user and specified date
public void createTimesheet(string userName, DateTime startDay)
{
    Timesheet newTimesheet = new Timesheet();
    var searchTs = from t in TimesheetDB.TimesheetList
                   where (t.worker.CompareTo(userName) == 0)
                   where t.periodStart == startDay
                   select t;
    try
    {
        //make sure the timesheet doesn't exist already
        if (searchTs.Count() == 0)
        {
            //Set pay period to start on Sunday 12:00am
            newTimesheet.periodStart = startDay;
            newTimesheet.periodEnd = startDay.AddDays(6.99999);
            newTimesheet.worker = userName;
            newTimesheet.approved = false;
            newTimesheet.locked = false;
            newTimesheet.submitted = false;

            //add timesheet and save to the database
            TimesheetDB.TimesheetList.Add(newTimesheet);
            TimesheetDB.Entry(newTimesheet).State = System.Data.EntityState.Added;
            TimesheetDB.SaveChanges();
        }
    }
    catch{}
}


//
//Retrieves and returns a specified user's timesheet for the specified date
public Timesheet getTimesheet(string user, DateTime tsDate)
{
    Timesheet resulttimesheet = new Timesheet();
```

```
    var searchTs = from m in TimesheetDB.TimesheetList
                   where (m.worker.CompareTo(user) == 0)
                   where m.periodStart <= tsDate
                   where m.periodEnd >= tsDate
                   select m;
    foreach (var item in searchTs)
    {
        resulttimesheet = item;
        return resulttimesheet;
    }
    return null;
}


//
/* Retrieves all workefforts from the database and sends them to the view,
 * along with a list of PCA Codes associated with each work effort
 */
[HttpGet]
public virtual ActionResult searchWorkEffort()
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        Authentication auth2 = new Authentication();
        if (auth2.isManager(this))
        {
            ViewBag.managerFlag = true;
        }

        var workEffortList = WorkEffortDB.WorkEffortList.ToList();
        //create a list of lists for pca codes
        //(each work effort will have a list of PCA codes)
        ViewBag.pcaListOfLists = new List<List<SelectListItem>>();
        foreach (var item in workEffortList)
        {
            ViewBag.pcaListOfLists.Add(getWePcaCodesSelectList(item));
        }

        return View(workEffortList);
    }
    else
    {
        return View("notLoggedOn");
    }
}


//
/* Retrieves the Work Effort object with the specified ID and sends it to the view,
 * along with a list of associated PCA Codes.
 */
[HttpGet]
public virtual ActionResult viewWorkEffort(int id)
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        Authentication newAuth = new Authentication();
        if (newAuth.isManager(this))
        {
            ViewBag.managerFlag = true;
        }

        WorkEffort workeffort = WorkEffortDB.WorkEffortList.Find(id);
        if (workeffort == null)
        {
            return HttpNotFound();
        }
        ViewBag.pcaList = getWePcaCodesSelectList(workeffort);
        ViewBag.WorkEffortID = workeffort.ID;
        return View(workeffort);
    }
    else
    {
        return View("notLoggedOn");
    }
}


//
/* Retrieves the hours object with the specified ID, changes the number of hours,
 * and saves the changes to the database
```

```
 */
[HttpPost]
public virtual bool editHours(int id, int numHours)
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        Hours tmpHours = HoursDB.HoursList.Find(id);
        tmpHours.hours = numHours;
        //save changes to database
        HoursDB.Entry(tmpHours).State = EntityState.Modified;
        HoursDB.SaveChanges();
        return true;
    }
    else
    {
        return false;
    }
}


//
/* Removes all the zero hours entries for the given workEffort/timeCode pair for the
 * week on the user's timesheet. This results in that workEffort/timeCode pair being
 * removed from the viewTimesheet View. Redirects to viewTimesheet().
*/
[HttpPost]
public virtual ActionResult timesheetRemoveWorkEffort(int sundayID)
{
    Hours sunHours = HoursDB.HoursList.Find(sundayID);
    DateTime sunday = sunHours.timestamp;

    var searchHours = from h in HoursDB.HoursList
                      where (h.creator.CompareTo(sunHours.creator) == 0)
                      where h.workEffortID == sunHours.workEffortID
                      where (h.description.CompareTo(sunHours.description) == 0)
                      where h.timestamp >= sunday
                      where h.timestamp < System.Data.Objects.EntityFunctions.AddDays(sunday, 7)
                      select h;
    foreach (var item in searchHours)
    {
        HoursDB.HoursList.Remove(item);
        HoursDB.Entry(item).State = System.Data.EntityState.Deleted;
    }
    HoursDB.SaveChanges();
    return RedirectToAction("viewTimesheet", new { tsDate = sunday.AddDays(1) });
}


//
//Returns TRUE if the specified timesheet is locked
public bool isTimesheetLocked(string worker,DateTime refDate)
{
    DateTime todaysDate = DateTime.Now;
    Timesheet tmpTimesheet = getTimesheet(worker, refDate);
    if ( (tmpTimesheet != null)&&(tmpTimesheet.locked == true) )
    {
        return true;
    }
    return false;
}


//
// Retrieves the hours object with the specified ID and deletes it from the database
[HttpGet]
public virtual ActionResult deleteHours(int id)
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        Hours hours = HoursDB.HoursList.Find(id);
        HoursDB.Entry(hours).State = EntityState.Deleted;
        HoursDB.SaveChanges();
        return RedirectToAction("viewTimesheet", new { tsDate = hours.timestamp });
    }
    else
    {
        return View("notLoggedOn");
    }
}
```

```
//
/* Duplicates all hours entries from previous week's timesheet and sets number of hours
 * to zero for all of them.
 */
[HttpGet]
public virtual ActionResult copyTimesheet()
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        string userName = User.Identity.Name;
        Timesheet previousTimesheet = new Timesheet();
        DateTime dayFromPrevPeriod = DateTime.Now;
        dayFromPrevPeriod = dayFromPrevPeriod.AddDays(-7);
        List<Hours> resultHours = new List<Hours>();

        //Select the timesheet from the previous pay period if it exists
        var searchTs = from m in TimesheetDB.TimesheetList
                       where (m.worker.CompareTo(userName) == 0)
                       where m.periodStart <= dayFromPrevPeriod
                       where m.periodEnd >= dayFromPrevPeriod
                       select m;
        foreach (var item in searchTs)
        {
            previousTimesheet = item;
        }
        //Iterate through each entry from previous week and duplicate it for this week
        var searchHours = from m in HoursDB.HoursList
                          where (m.creator.CompareTo(userName) == 0)
                          where m.timestamp >= previousTimesheet.periodStart
                          where m.timestamp <= previousTimesheet.periodEnd
                          select m;
        foreach (var item in searchHours)
        {
            resultHours.Add(item);
        }
        foreach (var copiedHours in resultHours)
        {
            copiedHours.hours = 0;
            copiedHours.timestamp = DateTime.Now.StartOfWeek(DayOfWeek.Sunday);
            addHours(copiedHours);
        }
        return RedirectToAction("viewTimesheet", new { tsDate = DateTime.Now });
    }
    else
    {
        return View("notLoggedOn");
    }
}


//
/* Retrieves a list of the current user's hours for the time period that tsDate falls within.
 * The list is saved in TempData[], then convertHoursForTimesheetView() is called.
 * convertHoursForTimesheetView() returns a list of TimesheetRow objects, which is sent to the view.
 */
public virtual ActionResult viewTimesheet(DateTime tsDate)
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        string userName = User.Identity.Name;
        checkForTimesheet(userName, tsDate);    //creates timesheet if it doesn't exist
        Timesheet timesheet = getTimesheet(userName, tsDate);
        Timesheet prevTimesheet = getTimesheet(userName, tsDate.AddDays(-7));

        ViewBag.timesheet = timesheet;
        //The View won't provide a link to previous timesheet unless it exists
        if (prevTimesheet == null)
        {
            ViewBag.noPreviousTimesheet = true;
        }
        //select all hours from the timesheet
        var hoursList = from m in HoursDB.HoursList
                        where (m.creator.CompareTo(userName) == 0)
                        where m.timestamp >= timesheet.periodStart
                        where m.timestamp <= timesheet.periodEnd
                        select m;

        TempData["hoursList"] = hoursList;
        //convert hoursList into a format that the view can use
        List<TimesheetRow> tsRows = convertHoursForTimesheetView();
```

```
        ViewBag.workEffortList = getVisibleWorkEffortSelectList(getUserDivision());
        return View(tsRows);
    }
    else
    {
        return View("notLoggedOn");
    }
}


//
/* Uses TempData["hoursList"] (assigned by the calling function) to create a list of objects
 * that each contain number of hours for Sun-Sat for each workEffort/timeCode pairing.  The
 * list of objects is then returned to the calling function
 */
public List<TimesheetRow> convertHoursForTimesheetView()
{
    WorkEffort effort = new WorkEffort();
    string effortDescription = "";
    List<WorkEffort> workEffortList = new List<WorkEffort>();
    List<string> timeCodeList = new List<string>();
    List<string> effortAndCodeConcat = new List<string>();
    List<TimesheetRow> tsRowList = new List<TimesheetRow>();

    //TempData["hoursList"] was assigned in viewTimesheet() before calling this method
    IEnumerable<Hours> hoursList = (IEnumerable<Hours>)TempData["hoursList"];

    //create a list of workEffort/timeCode pairings for the pay period
    foreach (var item in hoursList)
    {
        timeCodeList.Add(item.description);
        effort = WorkEffortDB.WorkEffortList.Find(item.workEffortID);
        if (effort != null)
        {
            workEffortList.Add(effort);
            effortAndCodeConcat.Add(effort.description + "::::" + item.description);
        }
    }
    //remove duplicates from the list
    effortAndCodeConcat = effortAndCodeConcat.Distinct().ToList();

    //for each unique workEffort/timeCode pairing
    foreach (var effortAndCode in effortAndCodeConcat)
    {
        TimesheetRow tmpTsRow = new TimesheetRow();

        /* save the work effort description and time code, then remove from the front of the list.
         * In the process, any duplicate pairs are ignored and removed by comparing to effortAndCodeConcat
         */
        for (int count = 0; count < 100; count++)
        {
            try
            {
                if ((effortAndCode.Contains(workEffortList.First().description)) &&
                     (effortAndCode.Contains(timeCodeList.First())))
                {
                    tmpTsRow.weID = workEffortList.First().ID;
                    tmpTsRow.workeffort = workEffortList.First().description;
                    tmpTsRow.timecode = timeCodeList.First();
                    workEffortList.RemoveAt(0);
                    timeCodeList.RemoveAt(0);
                    break;
                }
                else
                {
                    workEffortList.RemoveAt(0);
                    timeCodeList.RemoveAt(0);
                }
            }
            catch
            {
            }
        }

        //for each hours entry in the pay period
        foreach (var tmpVal in hoursList)
        {
            effortDescription = getWeDescription(tmpVal.workEffortID);
            //if the hours entry belongs to the unique workEffort/timeCode pairing
            if ((effortAndCode.CompareTo(effortDescription + "::::" + tmpVal.description) == 0))
            {
                switch (tmpVal.timestamp.DayOfWeek.ToString())
                {
```

```
                    case ("Sunday"):
                        tmpTsRow.sunHours = tmpVal;
                        break;
                    case ("Monday"):
                        tmpTsRow.monHours = tmpVal;
                        break;
                    case ("Tuesday"):
                        tmpTsRow.tueHours = tmpVal;
                        break;
                    case ("Wednesday"):
                        tmpTsRow.wedHours = tmpVal;
                        break;
                    case ("Thursday"):
                        tmpTsRow.thuHours = tmpVal;
                        break;
                    case ("Friday"):
                        tmpTsRow.friHours = tmpVal;
                        break;
                    case ("Saturday"):
                        tmpTsRow.satHours = tmpVal;
                        break;
                    default:
                        break;
                }
            }
        }
        //Add the TimesheetRow so it will be displayed in viewTimesheet View
        tsRowList.Add(tmpTsRow);
    }
    return tsRowList;
}


//
/* Retrieves and returns list of hours logged to the specified work effort during the specified
 * time frame by the specified user.
 */
public virtual ActionResult showWorkOnWorkEffort(int we, DateTime start, DateTime end, string userName)
{
    if (Request.IsAjaxRequest())
    {
        var hrsList = from h in HoursDB.HoursList
                      where (h.creator.CompareTo(userName) == 0)
                      where h.workEffortID == we
                      where h.timestamp >= start
                      where h.timestamp <= end
                      select h;
        return PartialView("_showWorkOnWorkEffort", hrsList.ToList());
    }
    return null;
}


//
/* Retrieves timesheet object with specified ID and changes submitted status to TRUE.
 * It then saves the change and redirects to viewTimesheet()
 */
public virtual ActionResult submitTimesheet(int id)
{
    if (id >= 0)
    {
        Authentication auth = new Authentication();
        if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
        {
            Timesheet ts = new Timesheet();
            ts = TimesheetDB.TimesheetList.Find(id);
            ts.submitted = true;
            //save changes to the database
            TimesheetDB.Entry(ts).State = System.Data.EntityState.Modified;
            TimesheetDB.SaveChanges();

            //send an email to employee to confirm timesheet submittal
            string body = "Your IDHW timesheet for the pay period of " + ts.periodStart +
                          " - " + ts.periodEnd + " has successfully been submitted.";
            SendEmail(ts.worker, "Timesheet Submitted", body);

            return RedirectToAction("viewTimesheet", new { tsDate = ts.periodStart.AddDays(2) });
        }
        else
        {
            return View("notLoggedOn");
        }
    }
```

```
    else
    {
        return View("error");
    }
}


//
/* Retrieves the specified employee's timesheet from the specified date and returns the
 * status as a string
 */
public virtual string getTimesheetStatus(string userName, DateTime refDate)
{
    Timesheet tmptimesheet = getTimesheet(userName, refDate);
    string status = "";

    if (tmptimesheet != null)
    {
        if (tmptimesheet.locked)
        {
            status = "locked";
        }
        else if (tmptimesheet.approved)
        {
            status = "approved";
        }
        else if (tmptimesheet.submitted)
        {
            status = "submitted";
        }
        else
        {
            status = "not submitted";
        }
    }
    return status;
}


//
//Returns the pay period for the reference date as a string
public virtual string getPayPeriod(DateTime refDate)
{
    DateTime startDay = refDate.StartOfWeek(DayOfWeek.Sunday);
    DateTime endDay = startDay.AddDays(6);
    string payPeriod = startDay.ToShortDateString() + " - " + endDay.ToShortDateString();
    return payPeriod;
}


//
// Retrieves the IDHW Divisions and returns as a list of strings
public virtual List<SelectListItem> getDivisionSelectList()
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        List<SelectListItem> divList = new List<SelectListItem>();
        var searchDivisions = from m in DivisionsDB.DivisionsList
                              select m;

        divList.Add(new SelectListItem { Text = "All", Value = "All" });

        foreach (var item in searchDivisions)
        {
            divList.Add(new SelectListItem
            {
                Text = item.divName,
                Value = item.divName
            });
        }
        return divList;
    }
    else
    {
        return null;
    }
}


//
/* Retrieves all PCA codes associated with the specified work effort and returns
 * them as a selection list
```

```
 */
public virtual List<SelectListItem> getWePcaCodesSelectList(WorkEffort we)
{
    List<SelectListItem> pcaList = new List<SelectListItem>();
    PcaCode tmpPca = new PcaCode();

    var searchPcaWe = from p in PCA_WEDB.PCA_WEList
                      where p.WE == we.ID
                      where p.active == true
                      select p;
    foreach (var item in searchPcaWe)
    {
        tmpPca = PcaCodeDB.PcaCodeList.Find(item.PCA);
        pcaList.Add(new SelectListItem
        {
            Text = tmpPca.code + " (" + tmpPca.division + ")",
            Value = tmpPca.code.ToString()
        });
    }
    return pcaList;
}


//
/* Retrieves all PCA codes associated with the specified work effort and
 * returns them in a string, separated by commas
 */
public virtual string getWePcaCodesString(int weID)
{
    PcaCode tmpPca = new PcaCode();
    string pcaString = "";
    WorkEffort we = WorkEffortDB.WorkEffortList.Find(weID);

    var searchPcaWe = from p in PCA_WEDB.PCA_WEList
                      where p.WE == we.ID
                      where p.active == true
                      select p;
    foreach (var item in searchPcaWe)
    {
        tmpPca = PcaCodeDB.PcaCodeList.Find(item.PCA);
        pcaString = pcaString + " " + tmpPca.code + ",";
    }
    pcaString = pcaString.TrimEnd(',');
    return pcaString;
}


//
/* Retrieves all Earnings Code objects, then creates and returns them as a list of
 * strings, each containing a concatenated earnings code and description.
 */
public virtual List<string> getTimeCodeList()
{
    List<string> timeCodesList = new List<string>();
    var searchEarnCodes = from m in EarningsCodesDB.EarningsCodesList
                          select m;
    timeCodesList.Add("--- Choose a Time Code ---");
    foreach (var item in searchEarnCodes)
    {
        timeCodesList.Add(item.earningsCode + " " + item.description);
    }
    return timeCodesList;
}


//
/* Retrieves all non-hidden Work Efforts within the specified division, then returns
 * them as a selection list. Since the division is not stored with a Work Effort, the
 * PCA Codes associated with each Work Effort instance must be retrieved.  If at
 * least one PCA Code is from the specified division, then the Work Effort is added
 * to the selection list.
 */
public virtual List<SelectListItem> getVisibleWorkEffortSelectList(string division)
{
    List<SelectListItem> effortList = new List<SelectListItem>();
    PcaCode tmpPca = new PcaCode();
    string tmpValue = "";

    var searchEfforts = from m in WorkEffortDB.WorkEffortList
                        select m;
    //narrow down to work efforts in the specified division
    //(PCA codes and PCA_WE must be used to get all work efforts in the division)
    foreach (var we in searchEfforts)
```

```
    {
        if (we.hidden != true)
        {
            var searchPcaWe = from p in PCA_WEDB.PCA_WEList
                              where p.WE == we.ID
                              where p.active == true
                              select p;
            foreach (var pca_we in searchPcaWe)
            {
                tmpPca = PcaCodeDB.PcaCodeList.Find(pca_we.PCA);
                //if the PCA is in the user's division, then add the Work Effort to the list
                if (tmpPca.division.CompareTo(division) == 0)
                {
                    tmpValue = we.ID.ToString();
                    effortList.Add(new SelectListItem
                    {
                        Text = we.description,
                        Value = tmpValue
                    });
                    break;
                }
            }
        }
    }
    return effortList;
}


//
/* Retrieves the Work Effort object with specified ID and returns the description of
 * as a string.  If the Work Effort has been deactivated, then a message is returned
 * that shows the ID and states that the Work Effort no longer exists.
 */
public virtual string getWeDescription(int id)
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        string weDescription = "";
        var searchWorkEfforts = from w in WorkEffortDB.WorkEffortList
                                where w.ID == id
                                select w;
        foreach (var item in searchWorkEfforts)
        {
            weDescription = item.description;
        }
        //If the Work Effort no longer exists, return the id and a message
        if (weDescription.Length == 0)
        {
            weDescription = "(Work Effort no longer exists. The unique ID was " + id + ")";
        }
        return weDescription;
    }
    else
    {
        return null;
    }
}


//
/* Retrieves Work Effort object with specified ID and returns it's time
 * boundaries as a string (note: it's called from addHours View)
 */
public string getWeTimeBoundsString(int id)
{
    WorkEffort we = WorkEffortDB.WorkEffortList.Find(id);
    string bounds = we.startDate + " - " + we.endDate;
    return bounds;
}


//
// Retrieves the division that the logged in user works for and returns it as a string
public virtual string getUserDivision()
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        string division = "";
        var searchUsers = from m in TARSUserDB.TARSUserList
                          where (m.userName.CompareTo(User.Identity.Name) == 0)
                          select m;
```

```
        foreach (var item in searchUsers)
        {
            division = item.company;
        }
        return division;
    }
    else
    {
        return null;
    }
}


//
// Retrieves the department that the logged in user works for and returns it as a string
public virtual string getUserDepartment()
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        string department = "";
        var searchUsers = from m in TARSUserDB.TARSUserList
                          where (m.userName.CompareTo(User.Identity.Name) == 0)
                          select m;
        foreach (var item in searchUsers)
        {
            department = item.department;
        }
        return department;
    }
    else
    {
        return null;
    }
}


//
/* Returns title that is shown when hovering over a timesheet day cell in
 * viewTimesheet view. If the timesheet is submitted or locked, then the title
 * will reflect that (unless user is an Admin).  Otherwise, it will show
 * "Add/Edit Hours"
 */
public virtual string getTimesheetDayCellTitle(Timesheet ts)
{
    string title = "";
    Authentication auth = new Authentication();

    if ( (ts.approved == true) || (ts.locked == true) )
    {
        if (auth.isAdmin(this))
        {
            title = "Admin: Add/Edit Hours";
        }
        else
        {
            title = getTimesheetStatus(ts.worker, ts.periodStart);
        }
    }
    else
    {
        title = "Add/Edit Hours";
    }
    return title;
}


//
/* Calls getVisibleWorkEffortSelectList(), then converts the result into a Json list
 * and returns it
 */
public ActionResult jsonWorkEffortSelectList(string division)
{
    List<SelectListItem> weSelectList = getVisibleWorkEffortSelectList(division);
    List<string> weIDList = new List<string>();
    foreach (var item in weSelectList)
    {
        weIDList.Add(item.Value);
    }
    return Json(weIDList.Select(x => new { value = x, text = getWeDescription(Convert.ToInt32(x)) }),
                JsonRequestBehavior.AllowGet
                );
}
```

```
        //
        // Calls getTimeCodeListList(), then converts the result into a Json list and returns it.
        public JsonResult jsonTimeCodeSelectList()
        {
            IEnumerable<string> weSelectList = getTimeCodeList();

            return Json(weSelectList.Select(x => new { value = x, text = x }),
                        JsonRequestBehavior.AllowGet
                        );
        }


        //
        // Sends a reminder email to all users who haven't submitted their timesheet by Saturday morning
        public void reminderToSubmitTimesheet()
        {
            DateTime refDate = DateTime.Now;
            string body = "Please submit your IDHW timesheet by the end of the day today.  <br /><br />Thanks!";
            var searchTimesheets = from t in TimesheetDB.TimesheetList
                                   where t.periodStart <= refDate
                                   where t.periodEnd >= refDate
                                   where t.submitted != true
                                   select t;
            foreach (var item in searchTimesheets)
            {
                SendEmail(item.worker, "Reminder to submit timesheet today", body);
            }
            return;
        }


        //
        //Sends an email from local server
        internal static void SendEmail(string userName, string subject, string body)
        {
string toAddress = "zeke_long@hotmail.com";
            //string toAddress = getEmailAddress(userName);
            try
            {
                MailMessage mailMessage = new MailMessage();
                mailMessage.To.Add(new MailAddress(toAddress));
                mailMessage.Subject = subject;
                mailMessage.Body = body;
                mailMessage.IsBodyHtml = true;

                var client = new SmtpClient();
                client.Send(mailMessage);
            }
            catch (Exception ex)
            {
                var tmpVar = ex;
            }
        }

    }
}
```

10.2 Manager Controller

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.Mvc;

using TARS.Helpers;
using TARS.Models;

namespace TARS.Controllers
{
    public class ManagerController : UserController
    {
        protected HistoryDBContext HistoryDB = new HistoryDBContext();

        //
        // Returns manager Index view
        // Overridden from User/Index, which was virtual.
        public override ActionResult Index()
        {
            Authentication auth = new Authentication();
            if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
            {
                return View();
            }
            else
            {
                return View("error");
            }
        }


        //
        /* Retrieves all PCA codes for the specified division and sends them to the view as
         * a list. The division name and a division selection list are also sent to the view.
         */
        public virtual ActionResult searchPCA(string division = null)
        {
            Authentication auth = new Authentication();
            if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
            {
                ViewBag.divisionList = getDivisionSelectList();
                if ( (division == null)||(division.CompareTo("All") == 0) )
                {
                    ViewBag.division = "All";
                    return View(PcaCodeDB.PcaCodeList.ToList());
                }
                else
                {
                    ViewBag.division = division;
                    var pcaList = from p in PcaCodeDB.PcaCodeList
                                  where (p.division.CompareTo(division) == 0)
                                  select p;
                    return View(pcaList.ToList());
                }
            }
            else
            {
                return View("error");
            }
        }


        //
        /* Retrieves all the employee objects of employees that work for the specified departement
         * within the Information Technology division (as per client request), then returns them
         * as a list. If department is null, it displays all employees in the division
         */
        public virtual ActionResult userManagement(DateTime refDate, string department = null)
        {
            Authentication auth = new Authentication();
            if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
            {
                //if it's a redirect from submitRejectTimesheet()
                if (TempData["emailSentFlag"] != null)
                {
                    ViewBag.emailSentFlag = true;
                    ViewBag.messageRecipient = TempData["recipient"];
                }
```

```
        string division = getUserDivision();
        IEnumerable<TARSUser> employees = getDivisionEmployeeObjList(division, department);
        ViewBag.division = division;
        ViewBag.departmentList = getDepartmentSelectList(division);
        ViewBag.refDate = refDate;
        ViewBag.refSunday = refDate.StartOfWeek(DayOfWeek.Sunday);
        ViewBag.refPayPeriod = getPayPeriod(refDate);
        return View(employees);
    }
    else
    {
        return View("error");
    }
}


//
/* Retrieves all Work Efforts and sends them to the view as a list, along with a
 * list of PCA Codes associated with each Work Effort. Also, sends variables that
 * the view uses to determine if error messages should be displayed
 */
[HttpGet]
public virtual ActionResult weManagement()
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        var workEffortList = WorkEffortDB.WorkEffortList.ToList();

        //create a list of lists for pca codes
        //(each work effort will have a list of PCA codes)
        ViewBag.pcaListOfLists = new List<List<SelectListItem>>();
        foreach (var item in workEffortList)
        {
            ViewBag.pcaListOfLists.Add(getWePcaCodesSelectList(item));
        }

        //check if an "unable to hide Work Effort error should be displayed"
        if (TempData["failedHide"] != null)
        {
            ViewBag.failedHide = true;
        }
        //check if an "unable to delete Work Effort error should be displayed"
        if (TempData["failedDelete"] != null)
        {
            ViewBag.failedDelete = true;
        }
        return View(workEffortList);
    }
    else
    {
        return View("error");
    }
}


//
//This was attached to delete; not sure what this is yet, but it doesn't explode!
protected override void Dispose(bool disposing)
{
    PcaCodeDB.Dispose();
    base.Dispose(disposing);
}


//
/* Creates an empty Work Effort object and sends it to the view, along with a
 * selection list of divisions.
 */
public virtual ActionResult addWorkEffort()
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        ViewBag.divisionList = getDivisionSelectList();
        return View(new WorkEffort());
    }
    else
    {
        return View("error");
    }
}
```

```
//
/* Receives a Work Effort object and checks to make sure that the dates are valid
 * and that they are within the associated PCA Code's time boundaries.  If so, then
 * it is saved to the database.  If not, then the object is returned to the view
 * for editing.
 */
[HttpPost]
public virtual ActionResult addWorkEffort(WorkEffort workeffort)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        if (ModelState.IsValid)
        {
            //make sure there is an end date
            if (workeffort.endDate == null)
            {
                workeffort.endDate = DateTime.MaxValue;
            }

            //make sure the start date is before the end date
            if (workeffort.startDate > workeffort.endDate)
            {
                ViewBag.divisionList = getDivisionSelectList();
                ViewBag.endBeforeStartFlag = true;
                return View(workeffort);
            }

            //make sure it falls within it's associated PCA code's time boundaries
            if (verifyWeTimeBounds(workeffort, workeffort.pcaCode) == true)
            {
                //update WorkEffort table in database
                WorkEffortDB.WorkEffortList.Add(workeffort);
                WorkEffortDB.SaveChanges();

                //add the PCA_WE association to PCA_WE table
                PCA_WE tmpPcaWe = new PCA_WE();
                tmpPcaWe.WE = workeffort.ID;
                tmpPcaWe.PCA = getPcaIdFromCode(workeffort.pcaCode);
                tmpPcaWe.associationStartDate = DateTime.Now;
                tmpPcaWe.associationEndDate = DateTime.MaxValue;
                tmpPcaWe.active = true;
                PCA_WEDB.PCA_WEList.Add(tmpPcaWe);
                PCA_WEDB.SaveChanges();

                return RedirectToAction("weManagement");
            }
            else
            {
                ViewBag.divisionList = getDivisionSelectList();
                ViewBag.notWithinTimeBounds = true;
                return View(workeffort);
            }
        }
        return View("error");
    }
    else
    {
        return View("error");
    }
}


//
/* Retrieves the Work Effort object with specified ID and returns it to the view.
 * If the user is an Admin, ViewBag.adminFlag is set to TRUE.  In the view, if
 * ViewBag.adminFlag is TRUE, then an option to ADD or REMOVE PCA Code associations
 * is displayed.
 */
[HttpGet]
public virtual ActionResult editWorkEffort(int id)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        WorkEffort workeffort = WorkEffortDB.WorkEffortList.Find(id);
        ViewBag.pcaList = getWePcaCodesSelectList(workeffort);

        Authentication newAuth = new Authentication();
        if (newAuth.isAdmin(this))
        {
            ViewBag.adminFlag = true;
```

```
        }
        return View(workeffort);
    }
    else
    {
        return View("error");
    }
}


//
/* Receives a Work Effort object and checks to make sure that the dates are valid
 * and that they are within the associated PCA Code's time boundaries.  If so, then
 * it is saved to the database.  If not, then the object is returned to the view
 * for editing.
 */
[HttpPost]
public virtual ActionResult editWorkEffort(WorkEffort workeffort)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        if (ModelState.IsValid)
        {
            //make sure the start date is before the end date
            if (workeffort.startDate > workeffort.endDate)
            {
                ViewBag.endBeforeStartFlag = true;
                ViewBag.pcaList = getWePcaCodesSelectList(workeffort);
                Authentication newAuth = new Authentication();
                if (newAuth.isAdmin(this))
                {
                    ViewBag.adminFlag = true;
                }
                return View(workeffort);
            }

            //make sure it falls within it's associated PCA code's time boundaries
            if (verifyWeTimeBounds(workeffort, workeffort.pcaCode) == true)
            {
                WorkEffortDB.WorkEffortList.Add(workeffort);
                WorkEffortDB.Entry(workeffort).State = System.Data.EntityState.Modified;
                WorkEffortDB.SaveChanges();
                return RedirectToAction("weManagement");
            }
            else
            {
                ViewBag.notWithinTimeBounds = true;
                ViewBag.pcaList = getWePcaCodesSelectList(workeffort);
                Authentication newAuth = new Authentication();
                if (newAuth.isAdmin(this))
                {
                    ViewBag.adminFlag = true;
                }
                return View(workeffort);
            }
        }
        return View(workeffort);
    }
    else
    {
        return View("error");
    }
}


//
/* Retrieves Work Effort object with specified ID and returns it to the view, along
 * with a selection list of PCA Codes associated with it.
 */
[HttpGet]
public virtual ActionResult deleteWorkEffort(int id)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        WorkEffort workeffort = WorkEffortDB.WorkEffortList.Find(id);
        ViewBag.pcaList = getWePcaCodesSelectList(workeffort);
        return View(workeffort);
    }
    else
    {
        return View("error");
```

```
    }
}


//
/* Receives a Work Effort ID and calls a function to check if there are any hours billed to
 * it.  If so, TempData["failedDelete"] is set to TRUE (so an error message will be displayed),
 * and it redirects back to "weManagement". Otherwise, the Work Effort object is retrieved and
 * deleted from the database, and deactivateAllPcaWeForWorkEffort(id) is called to deactivate
 * all of the PCA_We associations for the Work Effort.
 */
[HttpPost, ActionName("deleteWorkEffort")] //This action MUST match the above delete function.
public virtual ActionResult confirmedDeleteWorkEffort(int id)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        //make sure that there aren't any hours billed to the work effort
        if (checkWeForBilledHours(id) == true)
        {
            TempData["failedDelete"] = true;
            return RedirectToAction("weManagement");
        }
        else
        {
            WorkEffort workeffort = WorkEffortDB.WorkEffortList.Find(id);
            //change the active status to FALSE for all PCA_WE entries for the work effort
            deactivateAllPcaWeForWorkEffort(id);
            //delete the work effort
            WorkEffortDB.WorkEffortList.Remove(workeffort);
            WorkEffortDB.SaveChanges();
            return RedirectToAction("weManagement");
        }
    }
    else
    {
        return View("error");
    }
}


//
// Changes active status to FALSE for all PCA_WE entries for the specified Work Effort.
// Also changes "associationEndDate" to the current day and time
public void deactivateAllPcaWeForWorkEffort(int id)
{
    List<PCA_WE> tmpPcaWe = new List<PCA_WE>();
    var searchPcaWe = from p in PCA_WEDB.PCA_WEList
                      where p.WE == id
                      select p;
    tmpPcaWe.AddRange(searchPcaWe);
    foreach (var item in tmpPcaWe)
    {
        item.active = false;
        item.associationEndDate = DateTime.Now;
        //save changes in the database
        PCA_WEDB.Entry(item).State = System.Data.EntityState.Modified;
        PCA_WEDB.SaveChanges();
    }
    return;
}


//
/* Retrieves the Work Effort object with specified ID and checks to see if any
 * hours are currently billed to it in timesheets that are NOT locked yet.
 * Returns TRUE if there are.
 */
public bool checkWeForBilledHours(int id)
{
    Timesheet tmpTimesheet = new Timesheet();

    //get all hours that have been billed to the work effort
    var searchHours = from h in HoursDB.HoursList
                      where h.workEffortID == id
                      select h;
    foreach (var hrs in searchHours)
    {
        //If the corresponding timesheet is still active, return TRUE
        tmpTimesheet = getTimesheet(hrs.creator, hrs.timestamp);
        if (tmpTimesheet.locked == false)
        {
            return true;
```

```
        }
    }
    return false;       //if there are no hours billed to it, return false
}


//
/* Retrieves Work Effort object with specified ID, and changes "hidden" to TRUE
 * if the end date was at least one day ago.  (note: Hidden Work Efforts don't
 * show up for employee's when they are adding a Work Effort to their timesheet,
 * but they do still show up in Manager/weManagement where they can be edited and
 * un-hidden)
 */
public virtual ActionResult hideWorkEffort(int id)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        WorkEffort we = WorkEffortDB.WorkEffortList.Find(id);
        if (we.endDate < DateTime.Today)
        {
            we.hidden = true;
            WorkEffortDB.Entry(we).State = System.Data.EntityState.Modified;
            WorkEffortDB.SaveChanges();
            return RedirectToAction("WeManagement");
        }
        else
        {
            TempData["failedHide"] = true;
            return RedirectToAction("WeManagement");
        }
    }
    else
    {
        return View("error");
    }
}


//
/* Retrieves Work Effort object with specified ID, and changes "hidden" to FALSE
 * so users can log hours to it on their timesheet
 */
public virtual ActionResult unHideWorkEffort(int id)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        WorkEffort we = WorkEffortDB.WorkEffortList.Find(id);
        we.hidden = false;
        WorkEffortDB.Entry(we).State = System.Data.EntityState.Modified;
        WorkEffortDB.SaveChanges();

        return RedirectToAction("WeManagement");
    }
    else
    {
        return View("error");
    }
}


//
// Retrieves all PCA_WE objects and sends it to the view as a list
[HttpGet]
public virtual ActionResult searchPCA_WE()
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        return View(PCA_WEDB.PCA_WEList.ToList());
    }
    else
    {
        return View("error");
    }
}


//
// Retrieves PCA_WE object with specified ID and sends it to the view
[HttpGet]
public virtual ActionResult viewPCA_WE(int id)
```

```
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        PCA_WE pca_we = PCA_WEDB.PCA_WEList.Find(id);
        return View(pca_we);
    }
    else
    {
        return View("error");
    }
}


//
/* Retrieves a list of the specified user's hours for the time period that tsDate falls within.
 * The list is saved in TempData[], then convertHoursForTimesheetView() is called.
 * convertHoursForTimesheetView() returns a list of TimesheetRow objects, which is sent to the view.
 */
[HttpGet]
public virtual ActionResult approveTimesheet(int userKeyID, DateTime tsDate)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        TARSUser employee = TARSUserDB.TARSUserList.Find(userKeyID);
        Timesheet timesheet = getTimesheet(employee.userName, tsDate);

        if (timesheet == null)
        {
            createTimesheet(employee.userName, DateTime.Now);
            ViewBag.timesheet = getTimesheet(employee.userName, DateTime.Now);
        }
        else
        {
            ViewBag.timesheet = timesheet;
        }

        var hoursList = from m in HoursDB.HoursList
                        where (m.creator.CompareTo(employee.userName) == 0)
                        where m.timestamp >= timesheet.periodStart
                        where m.timestamp <= timesheet.periodEnd
                        select m;

        TempData["hoursList"] = hoursList;
        //convert hoursList into a format that the view can use
        List<TimesheetRow> tsRows = convertHoursForTimesheetView();

        ViewBag.workEffortList = getVisibleWorkEffortSelectList(getUserDivision());
        ViewBag.refDate = tsDate;
        ViewBag.userKeyID = userKeyID;
        return View(tsRows);
    }
    else
    {
        return View("error");
    }
}


//
/* Retrieves Timesheet object with specified ID and changes "submitted" and
 * "approved" statuses to TRUE.  Also sends a notification email to the employee
 */
public virtual ActionResult submitApproveTimesheet(int id)
{
    if (id >= 0)
    {
        Authentication auth = new Authentication();
        if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
        {
            Timesheet ts = new Timesheet();
            ts = TimesheetDB.TimesheetList.Find(id);
            ts.submitted = true;
            ts.approved = true;
            TimesheetDB.Entry(ts).State = System.Data.EntityState.Modified;
            //save changes to the database
            TimesheetDB.SaveChanges();

            //send an email to employee to notify of timesheet approval
            string body = "Your IDHW timesheet for the pay period of " + ts.periodStart +
                          " - " + ts.periodEnd + " has been approved by a manager.";
            SendEmail(ts.worker, "Timesheet Approved", body);
```

```
            return RedirectToAction("userManagement", new { refDate = DateTime.Now });
        }
        else
        {
            return View("error");
        }
    }
    else
    {
        return View("error");
    }
}


//
/* Retrieves Timesheet object with specified ID and changes "submitted" and
 * "approved" statuses to FALSE.  Also sends a notification email to the employee
 */
public virtual ActionResult submitRejectTimesheet(int id)
{
    if (id >= 0)
    {
        Authentication auth = new Authentication();
        if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
        {
            Timesheet ts = new Timesheet();
            ts = TimesheetDB.TimesheetList.Find(id);
            ts.submitted = false;
            ts.approved = false;
            //save changes to the database
            TimesheetDB.Entry(ts).State = System.Data.EntityState.Modified;
            TimesheetDB.SaveChanges();

            //send an email to employee to notify them
            string body = "Your IDHW timesheet for the pay period of " + ts.periodStart +
                          " - " + ts.periodEnd + " has been rejected by a manager. " +
                          "Please fix it and re-submit as soon as possible.<br /><br />Thanks!";
            SendEmail(ts.worker, "Rejected Timesheet", body);
            TempData["emailSentFlag"] = true;
            TempData["recipient"] = ts.worker;
            TempData["emailError"] = TempData["emailError"];

            return RedirectToAction("userManagement", new { refDate=DateTime.Now });
        }
        else
        {
            return View("error");
        }
    }
    else
    {
        return View("error");
    }
}


//
/* Retrieves Hours object with specified ID and sends it to the view, along with the
 * Work Effort that the hours are logged to (so description can be displayed), a
 * time code selection list (so the time code can be edited), and flags that let the
 * view know if the hours are editable.
 */
[HttpGet]
public virtual ActionResult managerEditHours(int hrsID, int tsID)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        Hours hours = HoursDB.HoursList.Find(hrsID);
        WorkEffort we = WorkEffortDB.WorkEffortList.Find(hours.workEffortID);
        Timesheet timesheet = TimesheetDB.TimesheetList.Find(tsID);
        ViewBag.timesheetLockedFlag = timesheet.locked;
        Authentication newAuth = new Authentication();
        bool adminFlag = newAuth.isAdmin(this);
        ViewBag.adminFlag = adminFlag;
        ViewBag.workEffort = we;
        ViewBag.timeCodeList = getTimeCodeList();
        return View(hours);
    }
    else
    {
        return View("error");
```

```
    }
}


//
/* Receives Hours object and saves it to the database as modified, then redirects
 * back to Manager/approveTimesheet
 */
[HttpPost]
public virtual ActionResult managerEditHours(Hours tmpHours)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        if (ModelState.IsValid)
        {
            HoursDB.Entry(tmpHours).State = EntityState.Modified;
            HoursDB.SaveChanges();
        }
        int userKeyID = getUserKeyID(tmpHours.creator);
        return RedirectToAction("approveTimesheet", new { userKeyID = userKeyID, tsDate = tmpHours.timestamp });
    }
    else
    {
        return View("error");
    }
}


//
/* Receives an employee's userName and retrieves the TARSUser object for that
 * userName, then returns the object's unique ID
 */
public int getUserKeyID(string worker)
{
    int userID = 0;
    var searchID = from m in TARSUserDB.TARSUserList
                   where (m.userName.CompareTo(worker) == 0)
                   select m;
    userID = searchID.First().ID;
    return userID;
}


//
/* Receives a 5-digit code and retrieves the PcaCode object with that code,
 * then returns the object's unique ID
 */
public int getPcaIdFromCode(int pcacode)
{
    int pcaID = 0;
    var searchID = from m in PcaCodeDB.PcaCodeList
                   where m.code == pcacode
                   select m;
    foreach (var item in searchID)
    {
        pcaID = item.ID;
    }
    return pcaID;
}


//
// Retrieves the PcaCode object with the specified code and returns the object
public PcaCode getPcaObjFromCode(int pcacode)
{
    PcaCode pcaCodeObj = new PcaCode();
    var searchPca = from m in PcaCodeDB.PcaCodeList
                    where m.code == pcacode
                    select m;
    foreach (var item in searchPca)
    {
        pcaCodeObj = item;
    }
    if (pcaCodeObj != null)
    {
        return pcaCodeObj;
    }
    else
    {
        return null;
    }
}
```

```
//
// Retrieves the pcaCode object with the specified ID and returns the 5-digit code
// (note: called from searchPCA_WE View)
public int getPcaCodeFromID(int id)
{
    PcaCode pcaCodeObj = PcaCodeDB.PcaCodeList.Find(id);
    if (pcaCodeObj != null)
    {
        return pcaCodeObj.code;
    }
    else
    {
        return 0;
    }
}


//
// Retrieves the PcaCode object with specified ID and returns the division name
// (note: called from searchPCA_WE View)
public string getPcaDivisionFromID(int id)
{
    PcaCode pcaCodeObj = PcaCodeDB.PcaCodeList.Find(id);
    if (pcaCodeObj != null)
    {
        return pcaCodeObj.division;
    }
    else
    {
        return null;
    }
}


//
// Retrieves the PcaCode object with specified ID and returns the description
// (note: called from searchPCA_WE View)
public string getPcaDescriptionFromID(int id)
{
    PcaCode pcaCodeObj = PcaCodeDB.PcaCodeList.Find(id);
    if (pcaCodeObj != null)
    {
        return pcaCodeObj.description;
    }
    else
    {
        return null;
    }
}


//
/* Retrieves all PCA Codes for the specified division and returns the result
 * as a selection list
 */
public virtual List<SelectListItem> getDivisionPcaCodeList(string division)
{
    List<SelectListItem> pcaCodesList = new List<SelectListItem>();
    var searchPcaCodes = from m in PcaCodeDB.PcaCodeList
                         select m;
    if (division.CompareTo("All") != 0)
    {
        searchPcaCodes = from m in searchPcaCodes
                         where (m.division.CompareTo(division) == 0)
                         select m;
    }
    foreach (var item in searchPcaCodes)
    {
        pcaCodesList.Add(new SelectListItem
        {
            Text = item.code.ToString() + " (" + item.division + ")",
            Value = item.code.ToString()
        });
    }
    return pcaCodesList;
}


//
/* Retrieves list of employees that work for specified division and department, and returns
 * them as a list.  If division is null, it returns a list of all employees
```

```
 */
public virtual List<TARSUser> getDivisionEmployeeObjList(string division = null, string department = null)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        List<TARSUser> divEmployees = new List<TARSUser>();
        var searchUsers = from m in TARSUserDB.TARSUserList
                          select m;
        if ( (division != null)&&(division.CompareTo("All") != 0) )
        {
            if ( (department != null)&&(department.CompareTo("All") != 0) )
            {
                searchUsers = from m in searchUsers
                              where (m.company.CompareTo(division) == 0)
                              where (m.department.CompareTo(department) == 0)
                              select m;
            }
            else
            {
                searchUsers = from m in searchUsers
                              where (m.company.CompareTo(division) == 0)
                              select m;
            }
        }
        foreach (var item in searchUsers)
        {
            divEmployees.Add(item);
        }
        return divEmployees;
    }
    else
    {
        return null;
    }
}


//
/* Retrieves list of employee that work for specified division and department, and returns
 * the names as a selection list.  If division is null, it returns all employee names
 */
public virtual List<SelectListItem> getDivisionEmployeeSelectList(string division = null)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        List<SelectListItem> employeeNames = new List<SelectListItem>();
        var searchUsers = from m in TARSUserDB.TARSUserList
                          select m;
        if ((division != null) && (division.CompareTo("All") != 0))
        {
            searchUsers = from m in searchUsers
                          where (m.company.CompareTo(division) == 0)
                          select m;
        }

        employeeNames.Add(new SelectListItem { Text = "All", Value = "All" });

        foreach (var item in searchUsers)
        {
            employeeNames.Add(new SelectListItem
            {
                Text = item.userName,
                Value = item.userName
            });
        }
        return employeeNames;
    }
    else
    {
        return null;
    }
}


//
/* Retrieves all departments that have at least one employee in the specified division
 * and returns them as a selection list
 */
public List<SelectListItem> getDepartmentSelectList(string division)
{
    List<SelectListItem> deptSelectList = new List<SelectListItem>();
```

```
    List<string> deptList = new List<string>();
    var searchUsers = from u in TARSUserDB.TARSUserList
                  where (u.company.CompareTo(division) == 0)
                  select u;
    //store list of departments in the division (with duplicates)
    foreach (var item in searchUsers)
    {
        deptList.Add(item.department);
    }
    //remove duplicates
    deptList = deptList.Distinct().ToList();

    deptSelectList.Add(new SelectListItem { Text = "All", Value = "All" });
    foreach (var item in deptList)
    {
        deptSelectList.Add(new SelectListItem
        {
            Text = item,
            Value = item
        });
    }
    return deptSelectList;
}


//
/* Calls getDivisionEmployeeSelectList(division) to get the selection list of employees
 * for the division, then converts it to a Json selection list and returns it
 */
public ActionResult jsonDivisionEmployeeSelectList(string division)
{
    List<SelectListItem> employeeSelectList = getDivisionEmployeeSelectList(division);
    List<string> nameList = new List<string>();
    foreach (var item in employeeSelectList)
    {
        nameList.Add(item.Value);
    }
    return Json(nameList.Select(x => new { value = x, text = x }),
                JsonRequestBehavior.AllowGet
                );
}


//
/* Checks to see if the work effort falls within the specified PCA code's time boundaries.
 * If so, it returns TRUE.
 */
public bool verifyWeTimeBounds(WorkEffort effort, int pca)
{
    var searchPCA = from m in PcaCodeDB.PcaCodeList
                  where (m.code.CompareTo(pca) == 0)
                  select m;
    foreach (var item in searchPCA)
    {
        if ((item.startDate <= effort.startDate) && (item.endDate >= effort.endDate))
        {
            return true;
        }
    }
    return false;
}


//
// Retrieves PcaCode object with specified code and returns its time boundaries as a string
//(note: it's called from addWorkEffort View)
public string getPcaTimeBoundsString(int pcacode)
{
    string bounds = "";
    var searchPCA = from m in PcaCodeDB.PcaCodeList
                   where m.code == pcacode
                   select m;
    foreach (var item in searchPCA)
    {
        bounds = item.startDate + " - " + item.endDate;
    }
    return bounds;
}


//
// Returns all database tables as a selection list
public virtual List<SelectListItem> getDbTableSelectList()
```

```
{
    Authentication auth = new Authentication();
    if (auth.isUser(this) || Authentication.DEBUG_bypassAuth)
    {
        List<SelectListItem> tableSelectList = new List<SelectListItem>();
        var tableList = new List<string>();
        tableList.Add("Timesheets");
        tableList.Add("Hours");
        tableList.Add("WorkEfforts");
        tableList.Add("PcaCodes");
        tableList.Add("PCA_WE");
        tableList.Add("TARSUsers");
        tableList.Add("Holidays");

        tableSelectList.Add(new SelectListItem { Text = "All", Value = "All" });

        foreach (var item in tableList)
        {
            tableSelectList.Add(new SelectListItem
            {
                Text = item,
                Value = item
            });
        }
        return tableSelectList;
    }
    else
    {
        return null;
    }
}


//
/* Retrieves all History objects within the specified time frame and sends them
 * to the view as a list
 */
[HttpGet]
public ActionResult viewHistory(DateTime start, DateTime end)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        var searchHist = from h in HistoryDB.HistoryList
                         where h.timestamp >= start
                         where h.timestamp <= end
                         select h;

        ViewBag.start = start.ToShortDateString();
        ViewBag.end = end.ToShortDateString();
        ViewBag.divisionList = getDivisionSelectList();
        ViewBag.employeeList = getDivisionEmployeeSelectList();
        ViewBag.dbtableList = getDbTableSelectList();
        return View(searchHist.ToList());
    }
    else
    {
        return View("error");
    }
}


//
/* Recieves parameters for refining the history results, and narrows down the list to satisfy
 * those parameters.  Then it sends the results back to the view.
 */
[HttpPost]
public ActionResult viewHistory(DateTime start, DateTime end, string division = null, string un = null, string dbtable = null)
{
    Authentication auth = new Authentication();
    if (auth.isManager(this) || Authentication.DEBUG_bypassAuth)
    {
        IEnumerable<History> searchHist = from h in HistoryDB.HistoryList
                                          where h.timestamp >= start
                                          where h.timestamp <= end
                                          select h;

        if ( (division != null)&&(division.CompareTo("All") != 0) )
        {
            IEnumerable<SelectListItem> names = getDivisionEmployeeSelectList(division);
            List<History> tmpList = new List<History>();
            foreach (var item in names)
            {
```

```
                var name = from h in searchHist
                           where (h.username.CompareTo(item.Value) == 0)
                           select h;
                foreach (var nameObj in name)
                {
                    tmpList.Add(nameObj);
                }
            }
            searchHist = (IEnumerable<History>)tmpList;
        }

        if ((un != null) && (un.CompareTo("All") != 0))
        {
            searchHist = from h in searchHist
                         where (h.username.CompareTo(un) == 0)
                         select h;
        }

        if ( (dbtable != null)&&(dbtable.CompareTo("All") != 0) )
        {
            searchHist = from h in searchHist
                         where (h.dbtable.CompareTo(dbtable) == 0)
                         select h;
        }

        ViewBag.start = start.ToShortDateString();
        ViewBag.end = end.ToShortDateString();
        ViewBag.divisionList = getDivisionSelectList();
        ViewBag.employeeList = getDivisionEmployeeSelectList(division);
        ViewBag.dbtableList = getDbTableSelectList();
        return View(searchHist.ToList());
    }
    else
    {
        return View("error");
    }
}


//
/* Calls getDivisionPcaCodeList(division) to get the selection list of PCA Codes
 * for the division, then converts it to a Json selection list and returns it
 */
public ActionResult jsonPcaSelectList(string division)
{
    IEnumerable<SelectListItem> pcaList = getDivisionPcaCodeList(division);

    return Json(pcaList.Select(x => new { value = x.Value, text = x.Text }),
                JsonRequestBehavior.AllowGet
                );
}


//
// Sends notification emails to managers and admin if a PCA or Work Effort Expires this week
public void notificationOfPcaOrWeExpiration()
{
    DateTime refStart = DateTime.Now.StartOfWeek(DayOfWeek.Sunday);
    DateTime refEnd = refStart.AddDays(7);
    DateTime tmpEndDate = DateTime.Now;
    string body = "";
    string pcas = "";
    var searchPca = from p in PcaCodeDB.PcaCodeList
                    where p.endDate >= refStart
                    where p.endDate < refEnd
                    select p;
    var searchWe = from w in WorkEffortDB.WorkEffortList
                   where w.endDate >= refStart
                   where w.endDate < refEnd
                   select w;
    var searchUsers = from t in TARSUserDB.TARSUserList
                      where t.permission > 1
                      select t;
    foreach (var person in searchUsers)
    {
        foreach (var item in searchPca)
        {
            tmpEndDate = (DateTime)item.endDate;
            body = "Notification of PCA expiration this week. <br /><br />" +
                   "PCA code: " + item.code + " (" + item.division + ") <br />" +
                   "Description: \"" + item.description + "\" <br />" +
                   "End Date: " + tmpEndDate.ToShortDateString();
            SendEmail(person.userName, "PCA expiration notification", body);
```

```
            }
            foreach (var item in searchWe)
            {
                tmpEndDate = (DateTime)item.endDate;
                pcas = getWePcaCodesString(item.ID);
                body = "Notification of Work Effort expiration this week. <br /><br />" +
                        "Work Effort description: \"" + item.description + "\" <br />" +
                        "Associated PCA code(s): " + pcas + "<br />End Date: " +
                        tmpEndDate.ToShortDateString();
                SendEmail(person.userName, "Work Effort expiration notification", body);
            }
        }
        return;
    }
  }
}
```

10.3 Admin Controller

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.Mvc;

using TARS.Helpers;
using TARS.Models;

namespace TARS.Controllers
{
    public class AdminController : ManagerController
    {
        //
        // Displays the default admin view
        [HttpGet]
        public override ActionResult Index()
        {
            Authentication auth = new Authentication();
            if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
            {
                return View();
            }
            else
            {
                return View("error");
            }
        }


        //
        /* Retrieves all PcaCode objects for the specified division and sends them to the
         * view as a list, along with a division selection list (for narrowing results)
         * and current division name.
         */
        public ActionResult maintainPCA(string division = null)
        {
            Authentication auth = new Authentication();
            if(auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
            {
                ViewBag.divisionList = getDivisionSelectList();

                if (TempData["failedPcaDelete"] != null)
                {
                    ViewBag.failedPcaDelete = true;
                }
                if ((division == null) || (division.CompareTo("All") == 0))
                {
                    ViewBag.division = "All";
                    return View(PcaCodeDB.PcaCodeList.ToList());
                }
                else
                {
                    ViewBag.division = division;
                    var pcaList = from p in PcaCodeDB.PcaCodeList
                                  where (p.division.CompareTo(division) == 0)
                                  select p;
                    return View(pcaList.ToList());
                }
            }
            else
            {
                return View("error");
            }
        }


        //
        /* Creates an empty PcaCode object and sends it to the view, along with a division
         * selection list (for specifying what division the new PCA Code will be in)
         */
        [HttpGet]
        public virtual ActionResult addPCA()
        {
            Authentication auth = new Authentication();
            if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
            {
                ViewBag.divisionList = getDivisionSelectList();
                return View(new PcaCode());
            }
            else
```

```
    {
        return View("error");
    }
}


//
/* Recieves a PcaCode object and saves it to the database.  If the start date is
 * before the end date, or if the the PCA Code already exists in the division
 * and has dates that overlap the dates of the new PCA Code, then an error flag
 * is set and the object is sent back to the view for editing.
 */
[HttpPost]
public virtual ActionResult addPCA(PcaCode pcacode)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        if (ModelState.IsValid)
        {
            if (pcacode.endDate == null)
            {
                pcacode.endDate = DateTime.MaxValue;
            }
            //make sure the start date is before the end date
            if (pcacode.startDate > pcacode.endDate)
            {
                ViewBag.endBeforeStartFlag = true;
                ViewBag.divisionList = getDivisionSelectList();
                return View(pcacode);
            }

            /* Make sure that the dates don't overlap if there is another PCA with the same
             * code in the same division
             */
            if (pcaCheckIfDuplicate(pcacode) == false)
            {
                PcaCodeDB.PcaCodeList.Add(pcacode);
                PcaCodeDB.SaveChanges();
                return RedirectToAction("maintainPCA");
            }
            else
            {
                ViewBag.duplicatePcaFlag = true;
                ViewBag.divisionList = getDivisionSelectList();
                return View(pcacode);
            }
        }
        return View(pcacode);
    }
    else
    {
        return View("error");
    }
}


//
/* Retrieves PcaCode object with specified ID and sends it to the view, along with
 * a division selection list (for changing the division that the PCA Code is in).
 */
[HttpGet]
public virtual ActionResult editPCA(int id)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        PcaCode pcacode = PcaCodeDB.PcaCodeList.Find(id);
        ViewBag.divisionList = getDivisionSelectList();
        return View(pcacode);
    }
    else
    {
        return View("error");
    }
}


//
/* Recieves a PcaCode object and saves it to the database.  If the start date is
 * before the end date, then an error flag is set and the object is sent back to
 * the view for editing.
 */
```

```
[HttpPost]
public virtual ActionResult editPCA(PcaCode pcacode)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        //make sure startDate is prior to endDate
        if (pcacode.startDate > pcacode.endDate)
        {
            ViewBag.endBeforeStartFlag = true;
            ViewBag.divisionList = getDivisionSelectList();
            return View(pcacode);
        }

        /* Make sure that the dates don't overlap if there is another PCA with the same
         * code in the same division
         */
        if (pcaCheckIfDuplicate(pcacode) == true)
        {
            ViewBag.duplicatePcaFlag = true;
            ViewBag.divisionList = getDivisionSelectList();
            return View(pcacode);
        }

        TempData["tmpPcaCode"] = pcacode;
        return RedirectToAction("confirmEditPCA");
    }
    else
    {
        return View("error");
    }
}


//
/* Sends TempData["tmpPcaCode"] (which was assigned in editPCA() before calling this
 * method) to the view, along with a division selection list.
 */
[HttpGet]
public virtual ActionResult confirmEditPCA()
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        ViewBag.divisionList = getDivisionSelectList();
        return View(TempData["tmpPcaCode"]);
    }
    else
    {
        return View("error");
    }
}


//
// Receives a PcaCode object and saves it to the database as modified
[HttpPost]
public virtual ActionResult confirmEditPCA(PcaCode pcacode)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {

        if (ModelState.IsValid)
        {
            PcaCodeDB.PcaCodeList.Add(pcacode);
            PcaCodeDB.Entry(pcacode).State = System.Data.EntityState.Modified;
            PcaCodeDB.SaveChanges();
        }
        return RedirectToAction("maintainPCA");
    }
    else
    {
        return View("error");
    }
}


//
// Retrieves PcaCode object with specified ID and sends it to the view
[HttpGet]
public virtual ActionResult deletePCA(int id)
{
```

```
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        PcaCode pcacode = PcaCodeDB.PcaCodeList.Find(id);
        return View(pcacode);
    }
    else
    {
        return View("error");
    }
}


//
/* Recieves a PcaCode object ID and retrieves the object, then checks to make sure
 * that no Work Efforts are attached to it before deleting it from the database.  If
 * there are Work Efforts attached, then a flag is set so an error message will be
 * displayed, and the Work Effort is not deleted.
 */
[HttpPost, ActionName("deletePCA")] //This action MUST match the above delete function.
public virtual ActionResult confirmDeletePCA(int id)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        //make sure there are no work efforts attached to the PCA
        if (checkPcaForAttachedWorkEfforts(id) == true)
        {
            TempData["failedPcaDelete"] = true;
            return RedirectToAction("maintainPCA");
        }
        else
        {
            PcaCode pcacode = PcaCodeDB.PcaCodeList.Find(id);
            PcaCodeDB.PcaCodeList.Remove(pcacode);
            PcaCodeDB.SaveChanges();
            return RedirectToAction("maintainPCA");
        }
    }
    else
    {
        return View("error");
    }
}


//
/* Receives a PcaCode object ID, then checks to see if any Work Efforts are currently
 * attached to it. It does so by retrieving all PCA_WE entries with a PCA field that
 * matches the ID, and an "active" field set to TRUE. Returns TRUE if any entries are
 * found.
 */
public bool checkPcaForAttachedWorkEfforts(int id)
{
    var searchPcaWe = from p in PCA_WEDB.PCA_WEList
                      where p.PCA == id
                      where p.active == true
                      select p;
    foreach (var item in searchPcaWe)
    {
        //return true if there is an entry in PCA_WE table with a matching PCA id
        if (item != null)
        {
            return true;
        }
        return false;
    }
    return false;
}


//
// Retrieves all TARSUser objects and sends them to the view as a list
public ActionResult userMaintanence()
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        return View(TARSUserDB.TARSUserList.ToList());
    }
    else
    {
        return View("error");
```

```
    }
}


//
// NOT YET IMPLEMENTED
public ActionResult addUser()
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        return View();
    }
    else
    {
        return View("error");
    }
}


//
// NOT YET IMPLEMETED
public ActionResult editTARSUSer()
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        return View();
    }
    else
    {
        return View("error");
    }
}


//
// NOT YET IMPLEMENTED
public ActionResult endDateTARSUSer()
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        return View();
    }
    else
    {
        return View("error");
    }
}


//
/* Receives a PcaCode object and checks if its 5-digit code already exists in the
 * division.  If so, it checks if the start and end dates overlap and returns TRUE
 * if they do, FALSE if they don't.
 */
public bool pcaCheckIfDuplicate(PcaCode pca)
{
    bool existsFlag = false;
    var searchPca = from p in PcaCodeDB.PcaCodeList
                    where p.ID != pca.ID
                    where p.code == pca.code
                    where (p.division.CompareTo(pca.division) == 0)
                    select p;
    foreach (var item in searchPca)
    {
        //if the date ranges overlap, then set flag to TRUE
        if ( (pca.startDate <= item.endDate)&&(item.startDate <= pca.endDate) )
        {
            existsFlag = true;
        }
    }
    return existsFlag;
}


//
/* Retrieves WorkEffort object with specified ID and sends the description and
 * ID to the view, along with a division selection list (For selecting the division
 * that the new PCA Code will be from. There is a PCA Code dropdown list in the
 * view that is populated via jQuery when a division is selected)
 */
```

```
[HttpGet]
public virtual ActionResult addPCA_WE(int weID)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        WorkEffort we = WorkEffortDB.WorkEffortList.Find(weID);
        ViewBag.divisionList = getDivisionSelectList();
        ViewBag.workEffortDescription = we.description;
        ViewBag.workEffortId = weID;
        return View();
    }
    else
    {
        return View("error");
    }
}


//
/* Receives a PCA_WE object and saves it to the database if the Work Effort is within the
 * time bounds of the PCA Code.  If the Work Effort is NOT within the time PCA Code's time
 * bounds, a flag is set to display an error message, and the object is sent back to the
 * view for editing.
 */
[HttpPost]
public virtual ActionResult addPCA_WE(PCA_WE pca_we)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        //The view actually passed the PCA code instead of the ID, so set the ID to the correct value
        pca_we.PCA = getPcaIdFromCode(pca_we.PCA);

        WorkEffort effort = WorkEffortDB.WorkEffortList.Find(pca_we.WE);
        ViewBag.workEffortDescription = effort.description;
        PcaCode pcaObj = PcaCodeDB.PcaCodeList.Find(pca_we.PCA);

        //make sure it falls within it's associated PCA code's time boundaries
        if (verifyWeTimeBounds(effort, pcaObj.code) == true)
        {
            //Make sure it's not a duplicate entry before adding to database
            if (checkIfDuplicatePcaWe(pca_we) == false)
            {
                //update PCA_WE table
                PCA_WEDB.PCA_WEList.Add(pca_we);
                PCA_WEDB.Entry(pca_we).State = System.Data.EntityState.Added;
                PCA_WEDB.SaveChanges();
            }
            return RedirectToAction("editWorkEffort", "Manager", new { id = pca_we.WE });
        }
        ViewBag.divisionList = getDivisionSelectList();
        ViewBag.workEffortId = effort.ID;
        ViewBag.outOfPcaTimeBounds = true;
        return View(pca_we);
    }
    else
    {
        return View("error");
    }
}


//
/* Retrieves WorkEffort object for the specified ID and sends the description and ID
 * to the view, along with a list of PCA Codes that are associated with the Work Effort
 */
[HttpGet]
public virtual ActionResult deletePCA_WE(int weID)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        WorkEffort we = WorkEffortDB.WorkEffortList.Find(weID);
        ViewBag.workEffortDescription = we.description;
        ViewBag.workEffortId = weID;
        ViewBag.pcaList = getWePcaCodesSelectList(we);
        return View();
    }
    else
    {
        return View("error");
    }
```

```
}


//
/* Receives a PCA_WE object and sets the "active" field to FALSE in the database if
 * the PCA Code is NOT the only PCA Code associated with the Work Effort.  If it is
 * the only remaining PCA Code associated with the Work Effort, then a flag is set to
 * display an error and the object is sent back to the view.
 * Deactivates the PCA_WE object and changes the endDate to the current day
 */
[HttpPost]
public virtual ActionResult deletePCA_WE(PCA_WE pca_we)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        //The view actually passed the PCA code instead of the ID, so set the ID to the correct value
        pca_we.PCA = getPcaIdFromCode(pca_we.PCA);
        int count = 0;

        PCA_WE tmpPcaWe = new PCA_WE();
        var searchPcaWe = from p in PCA_WEDB.PCA_WEList
                          where p.WE == pca_we.WE
                          where p.active == true
                          select p;
        foreach (var item in searchPcaWe)
        {
            //This if-statement will only be true once
            if (item.PCA == pca_we.PCA)
            {
                tmpPcaWe = PCA_WEDB.PCA_WEList.Find(item.ID);
            }
            count++;
        }

        if (count > 1)
        {
            //deactivate and set the end date for the PCA_WE entry
            tmpPcaWe.active = false;
            tmpPcaWe.associationEndDate = DateTime.Now;
            // save changes in database
            PCA_WEDB.Entry(tmpPcaWe).State = System.Data.EntityState.Modified;
            PCA_WEDB.SaveChanges();
            return RedirectToAction("weManagement", "Manager");
        }

        ViewBag.lastPcaFlag = true;
        WorkEffort we = WorkEffortDB.WorkEffortList.Find(pca_we.WE);
        ViewBag.workEffortDescription = we.description;
        ViewBag.workEffortId = we.ID;
        ViewBag.pcaList = getWePcaCodesSelectList(we);
        return View();
    }
    else
    {
        return View("error");
    }
}


//
/* Receives a PCA_WE object and checks if the association already exists in the
 * database. Returns TRUE if does. (note: Called by addPCA_WE() before saving the
 * PCA_WE object)
 */
public bool checkIfDuplicatePcaWe(PCA_WE pcawe)
{
    var searchPcaWe = from p in PCA_WEDB.PCA_WEList
                      where p.WE == pcawe.WE
                      where p.PCA == pcawe.PCA
                      where p.active == true
                      select p;
    foreach (var item in searchPcaWe)
    {
        if (item != null)
        {
            return true;
        }
    }
    return false;
}
```

```
//
/* Retrieves all the PCA codes in TARS and returns their 5-digit codes
 * as a list of strings.
 */
public virtual List<string> getAllPcaCodes()
{
    List<string> pcaList = new List<string>();
    string tmpPca = "";
    var searchPca = from m in PcaCodeDB.PcaCodeList
                    select m;
    foreach (var item in searchPca)
    {
        tmpPca = item.code.ToString();
        pcaList.Add(tmpPca);
    }
    return pcaList;
}


//
/* Retrieves all holidays that are within the same pay period as the specified
 * reference date. If the search results in an empty list, then FALSE is returned.
 * If there is at least one holiday in the list, then TRUE is returned.
 */
public bool isHolidayWeek(DateTime refDate)
{
    DateTime refStart = refDate.StartOfWeek(DayOfWeek.Sunday);
    DateTime refEnd = refStart.AddDays(7);
    var searchHolidays = from h in HolidaysDB.HolidaysList
                         where h.date >= refStart
                         where h.date < refEnd
                         select h;
    foreach (var item in searchHolidays)
    {
        return true;
    }
    return false;
}


//
/* Locks all timesheets that ended more than two days ago.  If there was a holiday during
 * the previous pay period, lockDate will have an extra day added to it.
 * (note: called from /ScheduledJobs/TarsScheduledJobs every Tuesday and Wednesday at 12am)
 */
public void lockTimesheets()
{
    DateTime refDate = DateTime.Now.Date;
    DateTime lockDate = DateTime.Now.Date;
    Timesheet tmpTimesheet = new Timesheet();
    List<Timesheet> tsList = new List<Timesheet>();

    // If there was a holiday, allow three days after periodEnd before locking
    if (isHolidayWeek(refDate.AddDays(-7)))
    {
        lockDate = refDate.AddDays(-1);
    }
    else
    {
        lockDate = refDate.AddDays(-2);
    }
    var searchTimesheets = from t in TimesheetDB.TimesheetList
                           where t.locked != true
                           where t.periodEnd < lockDate
                           select t;
    foreach (var item in searchTimesheets)
    {
        tmpTimesheet = item;
        tmpTimesheet.locked = true;
        tsList.Add(tmpTimesheet);
    }
    foreach (var item in tsList)
    {
        //save changes in database
        TimesheetDB.Entry(item).State = System.Data.EntityState.Modified;
        TimesheetDB.SaveChanges();
    }
}


//
/* Retrieves all TARSUser objects of employees that work for the specified departement
 * within the Information Technology division and sends them to the view as a list. If
```

```
 * department is null, it retrieves all employees in the division.
 */
public ActionResult viewTimesheetStatuses(DateTime refDate, string department = null)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        string division = getUserDivision();
        IEnumerable<TARSUser> employees = getDivisionEmployeeObjList(division, department);
        ViewBag.division = division;
        ViewBag.departmentList = getDepartmentSelectList(division);
        ViewBag.refDate = refDate;
        ViewBag.refSunday = refDate.StartOfWeek(DayOfWeek.Sunday);
        ViewBag.refPayPeriod = getPayPeriod(refDate);
        return View(employees);
    }
    else
    {
        return View("error");
    }
}


//
/* Retrieves the Timesheet object for the specified employee and reference date,
 * changes "locked", "approved", and "submitted" fields to FALSE, and saves the
 * changes in the database.
 */
public void adminUnlockTimesheet(string username, DateTime refDate)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        Timesheet tmpTs = new Timesheet();
        var searchTs = from t in TimesheetDB.TimesheetList
                       where (t.worker.CompareTo(username) == 0)
                       where t.periodStart <= refDate
                       where t.periodEnd >= refDate
                       select t;
        foreach (var item in searchTs)
        {
            tmpTs = item;
            tmpTs.locked = false;
            tmpTs.approved = false;
            tmpTs.submitted = false;
        }
        //save changes in database
        TimesheetDB.Entry(tmpTs).State = System.Data.EntityState.Modified;
        TimesheetDB.SaveChanges();
    }
    return;
}


//
// Retrieves all of the Holidays objects and sends them to the view.
public ActionResult viewHolidays()
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        List<Holidays> holidays = HolidaysDB.HolidaysList.ToList();
        return View(holidays);
    }
    else
    {
        return View("error");
    }
}


//
// Creates an empty Holidays object and sends it to the view.
[HttpGet]
public virtual ActionResult addHoliday()
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        return View(new Holidays());
    }
    else
    {
```

```
        return View("error");
    }
}


//
// Receives a Holidays object and saves it to the database
[HttpPost]
public virtual ActionResult addHoliday(Holidays holiday)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        HolidaysDB.HolidaysList.Add(holiday);
        HolidaysDB.SaveChanges();
        return RedirectToAction("viewHolidays");
    }
    else
    {
        return View("error");
    }
}


//
// Retrieves the Holidays object with specified ID and sends it to the view
[HttpGet]
public virtual ActionResult editHoliday(int id)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        Holidays holiday = HolidaysDB.HolidaysList.Find(id);
        return View(holiday);
    }
    else
    {
        return View("error");
    }
}


//
// Receives a Holidays object and saves it to the database as modified
[HttpPost]
public virtual ActionResult editHoliday(Holidays holiday)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        HolidaysDB.Entry(holiday).State = System.Data.EntityState.Modified;
        HolidaysDB.SaveChanges();
        return RedirectToAction("viewHolidays");
    }
    else
    {
        return View("error");
    }
}


//
// Retrieves the Holidays object with specified ID and sends it to the view
[HttpGet]
public virtual ActionResult deleteHoliday(int id)
{
    Authentication auth = new Authentication();
    if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
    {
        Holidays holiday = HolidaysDB.HolidaysList.Find(id);
        return View(holiday);
    }
    else
    {
        return View("error");
    }
}


//
/* Retrieves the Holidays object with specified ID and deletes if from the database
 * once the user confirms that they want to delete it
 */
```

```
        [HttpPost, ActionName("deleteHoliday")] //This action MUST match the above delete function.
        public virtual ActionResult confirmDeleteHoliday(int id)
        {
            Authentication auth = new Authentication();
            if (auth.isAdmin(this) || Authentication.DEBUG_bypassAuth)
            {
                Holidays holiday = HolidaysDB.HolidaysList.Find(id);
                HolidaysDB.HolidaysList.Remove(holiday);
                HolidaysDB.SaveChanges();
                return RedirectToAction("viewHolidays");
            }
            else
            {
                return View("error");
            }
        }

    }
}
```

11 Appendix D: Reference Links

Unit Testing ASP.NET MVC
http://msdn.microsoft.com/en-us/magazine/dd942838.aspx

SQL2008 integration w/ IIS7 MVC reference
http://blog.evonet.com.au/post/Setting-up-SQL-Server-2008-for-an-ASPNET-website-on-IIS-70.aspx

SQL2008 St15udio Manager
http://www.microsoft.com/download/en/details.aspx?id=7593

SQL2008 Server
http://www.microsoft.com/download/en/details.aspx?id=1695

Apache Directory Server (LDAP stand-in for Active Directory)
http://directory.apache.org/apacheds/1.5/

Apache Directory Studio (Essential Client for Managing ApacheDS)
http://directory.apache.org/studio/

Write an LDAP interface in C#
http://www.youcanlearnseries.com/programming%20Tips/CSharp/LDAPReader.aspx