

Spring 2010

Develop Best Practices for Designing Internal Business Database-Driven Web Applications

Stephen C. Rash
Regis University

Follow this and additional works at: <https://epublications.regis.edu/theses>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Rash, Stephen C., "Develop Best Practices for Designing Internal Business Database-Driven Web Applications" (2010). *All Regis University Theses*. 128.
<https://epublications.regis.edu/theses/128>

This Thesis - Open Access is brought to you for free and open access by ePublications at Regis University. It has been accepted for inclusion in All Regis University Theses by an authorized administrator of ePublications at Regis University. For more information, please contact epublications@regis.edu.

Regis University
College for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

Use of the materials available in the Regis University Thesis Collection ("Collection") is limited and restricted to those users who agree to comply with the following terms of use. Regis University reserves the right to deny access to the Collection to any person who violates these terms of use or who seeks to or does alter, avoid or supersede the functional conditions, restrictions and limitations of the Collection.

The site may be used only for lawful purposes. The user is solely responsible for knowing and adhering to any and all applicable laws, rules, and regulations relating or pertaining to use of the Collection.

All content in this Collection is owned by and subject to the exclusive control of Regis University and the authors of the materials. It is available only for research purposes and may not be used in violation of copyright laws or for unlawful purposes. The materials may not be downloaded in whole or in part without permission of the copyright holder or as otherwise authorized in the "fair use" standards of the U.S. copyright laws and regulations.

DEVELOP BEST PRACTICES FOR DESIGNING INTERNAL BUSINESS DATABASE-
DRIVEN WEB APPLICATIONS

A THESIS

SUBMITTED ON 21ST OF FEBRUARY, 2010

TO THE DEPARTMENT OF INFORMATION SYSTEMS
OF THE SCHOOL OF COMPUTER & INFORMATION SCIENCES
OF REGIS UNIVERSITY

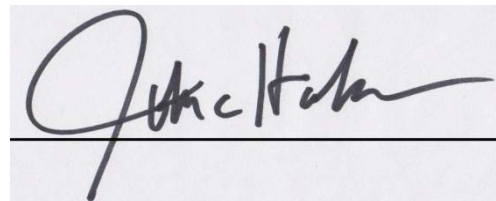
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF MASTER OF SCIENCE IN
SOFTWARE ENGINEERING

BY

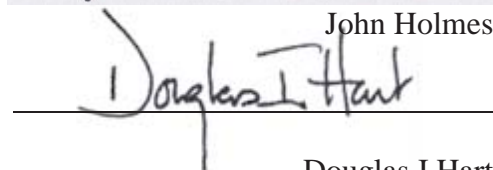


Stephen C. Rash

APPROVALS



John Holmes



Douglas I Hart



Shari Plantz-Masters

Abstract

When developing using newer technology, it is important for smaller information technology organizations to have universally accepted set of best practices to be able to successfully complete that type of endeavor. How can these universally accepted set of best practices be developed? Conducting research on accepted best practices can build the basis for your theories and assumptions. Next, in the context of your applications, develop an example application in the newer technology to test your theories and assumptions. Build the application like a construction project, the initial design is the blueprint, the database is the foundation and the user interface is the actual building. When you get right down to it, the principals of simplicity, consistency and user interaction are always best practices in developing applications.

Acknowledgements

To my wife, Melanie, thank you for giving me the support and encouragement to complete my degree. I could not have done this without you.

To my children, Carly, Patrick and Kevin, thank you for giving me the time and space I needed to complete my homework and even putting up with me when I was frustrated, but, most of all, understanding that this was something I needed to do.

To my parents, Chuck and Mary Kay, thank you both for the assistance and encouragement to strive for goals like this my entire life.

To my employer, thank you for the financial assistance to get this degree.

To the faculty and staff of Regis University, thank you all for making this learning experience wonderful for me, for helping me, challenging me and giving me the tools for continued success in college and out in the workforce.

Table of Contents

Abstract.....	ii
Acknowledgements.....	iii
Table of Contents.....	iv
List of Figures.....	vii
List of Tables.....	x
Chapter 1 – Introduction.....	1
Chapter 2 – Review of Literature and Research.....	3
Chapter 3 – Methodology.....	7
Chapter 4 – Project Analysis and Results.....	8
Initial Design.....	8
User Interaction.....	9
Requirements Gathering.....	10
Source of requirements.....	10
Defining use cases.....	12
Defining data elements.....	14
Defining visual aids.....	14
Documentation.....	16
Producing the scope document.....	16
Producing the design document.....	16
User design acceptance and signoff.....	17

Security Design 17

 Network Setup 17

 Database Setup..... 19

 Application Setup..... 19

Database Design 20

 Database Users 20

 Tables 21

 Logging 22

 Stored Procedures 25

 The lookup stored procedure. 26

 The select stored procedure. 28

 The modify stored procedure. 28

 Functions 28

 The Code Table..... 29

Application Design..... 29

 Layered (n-tier) Architecture 31

 Domain layer..... 31

 Service layer..... 31

 Business layer. 33

 Presentation layer..... 33

 Design Patterns 34

The Model.....	37
Domain Classes.....	37
Service Classes.....	37
Business Classes.....	44
The Controller.....	47
Controller Classes (Code Behind).....	47
Utility Classes.....	54
The View.....	55
User Controls.....	56
Cascading Style Sheets.....	62
Validations.....	64
Messaging.....	65
Popups.....	66
The Configuration File.....	69
Scheduled Tasks.....	70
Reporting.....	71
Chapter 5 – Project History.....	72
Chapter 6 – Conclusions.....	74
References.....	74

List of Figures

<i>Figure 1: Use Case Diagram</i>	13
<i>Figure 2: Use Case Narrative</i>	13
<i>Figure 3: Data Elements Spreadsheet</i>	14
<i>Figure 4: Use Case Visual Aid</i>	15
<i>Figure 5: Marked-up Use Case Visual Aid</i>	15
<i>Figure 6: Multiple Table Security Setup</i>	19
<i>Figure 7: Application Security Setup</i>	20
<i>Figure 8: Data Table Structure</i>	22
<i>Figure 9: Log Table Structure</i>	23
<i>Figure 10: Logging Trigger Structure</i>	24
<i>Figure 11: Lookup Stored Procedure</i>	25
<i>Figure 12: Select Stored Procedure</i>	26
<i>Figure 13: Modify Stored Procedure</i>	27
<i>Figure 14: Scalar Function</i>	28
<i>Figure 15: Layered Architecture</i>	31
<i>Figure 16: Domain Layer Structure</i>	32
<i>Figure 17: Service Layer Structure</i>	32
<i>Figure 18: Business Layer Structure</i>	33
<i>Figure 19: Presentation Layer Structure</i>	34
<i>Figure 20: Application Design Patterns</i>	36
<i>Figure 21: Domain Class</i>	38
<i>Figure 22: Factory Class</i>	39

Figure 23: IService (Marker Interface) Class. 40

Figure 24: Service Interface Class...... 40

Figure 25: Service SQL Implementation Class – Modify Function...... 41

Figure 26: Service SQL Implementation Class – Select Function. 42

Figure 27: Service SQL Implementation Class – Lookup Function. 43

Figure 28: Manager Class...... 45

Figure 29: IManager (Marker Interface) Class...... 45

Figure 30: Manager Interface Class. 46

Figure 31: Manager Rules Implementation Class. 47

Figure 32: Controller Class – Page Load. 48

Figure 33: Controller Class – Security. 49

Figure 34: Controller Class – New Functions. 49

Figure 35: Controller Class – Get Functions...... 51

Figure 36: Controller Class – Set Functions. 51

Figure 37: Controller Class – Set View State...... 52

Figure 38: Controller Class – Processing Functions. 53

Figure 39: Controller Class – Drop-down Lists. 54

Figure 40: Controller Class – Search Grid. 55

Figure 41: Controller Class – Messages...... 56

Figure 42: Controller Class – Buttons...... 56

Figure 43: Utility Class. 57

Figure 44: View – Search Criteria Fields. 58

Figure 45: View – Search Actions...... 58

<i>Figure 46: View – Search Grid</i>	59
<i>Figure 47: View – Data Actions</i>	60
<i>Figure 48: View – Validation Summary</i>	60
<i>Figure 49: View – Data Fields</i>	61
<i>Figure 50: View – Hidden Fields</i>	61
<i>Figure 51: View – User Display</i>	62
<i>Figure 52: Cascading Style Sheet</i>	63
<i>Figure 53: Changing a Cascading Style Sheet – Before</i>	63
<i>Figure 54: Changing a Cascading Style Sheet – After</i>	63
<i>Figure 55: Validation Code</i>	64
<i>Figure 56: Validation Display</i>	64
<i>Figure 57: Messaging Code</i>	65
<i>Figure 58: Messaging Display</i>	66
<i>Figure 59: Popup Cascading Style Sheet</i>	67
<i>Figure 60: Popup View Code</i>	67
<i>Figure 61: Popup Controller Code</i>	68
<i>Figure 62: Popup Display</i>	68
<i>Figure 63: Configuration File – Application Settings</i>	69
<i>Figure 64: Configuration File – Connection Strings</i>	70
<i>Figure 65: Configuration File – Authentication Settings</i>	70

List of Tables

Table 1: An analysis of the pros and cons of the different sources of requirements. 11

Table 2: An analysis of the pros and cons of the different security architectures. 18

Table 3: An analysis of the pros and cons of the different list storage architectures. 29

Table 4: An analysis of the different design patterns. 35

Table 5: Project history timeline..... 73

Chapter 1 – Introduction

Small Information Technology (IT) organizations that have recently attempted to develop database driven, internal web-based business applications to replace outdated windows-based applications have been unsuccessful. These organizations were unsuccessful because they do not have a universally accepted set of best practices to use for such development.

Hager, Kibler and Zach (1999) state that those who have used a web application have seen how “this world-changing technology... is burgeoning” and that “many managers are struggling with the various ways the concepts and technology can be leveraged to create corporate Intranets”.

“Companies wanted a way to centrally serve [applications], so some started to use Citrix Metaframes... which were essentially client-server systems with the client running on remote machines” (Hice, 2008, p. 20). Organizations are moving away from terminal servers (Citrix) and windows-based applications to internal web applications. Hice (2008) goes on to say that web applications “are the wave of the future” and that major software vendors are moving toward web-based systems. Organizations have found that they do not possess the knowledge base to successfully implement web applications; they are structured differently than old windows-based applications. Some organizations contract with outside web application developers to develop systems to help them gain the knowledge of the best practices to design and implement these types of applications on their own. Unfortunately, even the contractors do not always possess the knowledge of the best practices; there is often very little consistency in their approach and methodology. Organizations often cannot take anything the developers create and apply it to designing and developing new applications internally.

The goal of this research is to identify the best practices for developing and implementing new business applications by creating an example internal web-based application for organizations to understand and implement the best practices. The focus of this research is narrow in the context of understanding best practices for a basic business web application with a database data repository, any narrower, and the research would lose the validity of a real world problem.

Chapter 2 – Review of Literature and Research

The most important step in developing an internal business web-based application is the actual design. Davidson (2007) said it best:

...would you hire a contractor to build a house and then demand that they start pouring a foundation the very next day? Even worse, would you demand that it be done without blueprints or house plans? Hopefully, you answered "no" to both of these. A design is needed make sure that the house you *want* gets built, and that the land you are building it on will not sink into some underground cavern. If you answered yes, I am not sure if anything I can say will help you. (¶ 6)

A good, well thought out foundation can make or break the project. The design process cannot exist without the most important participants, the system users themselves. It is very important to get the users involved early and often. Hager, Kibler and Zach (1999) highlight the need to include the users:

User-centered design (UCD) is a technique for designing interfaces... that includes continuous and early focus on the users' tasks and goals. It is the best way to get potential end users to participate in the designing the interface, leveraging their specific knowledge as part of the overall process. (p. 58)

Meyers (2004) puts forth an interesting statement about the most important design guideline; "Make interfaces easy to use correctly and hard to use incorrectly". He also states that "if a user makes a mistake when using your interface, *it's your fault*" (Meyers, 2004, p. 14). Understanding how a user will use a system can assist an application developer in designing a system the user

will actually use successfully. “[D]esigners need to train themselves to anticipate what clients might reasonably like to do, and then facilitate that activity” (Meyers, 2004, p. 16).

Design patterns can be very useful in many aspects of designing an application. Fowler (2003) says that “patterns are half-baked – meaning you always have to finish them yourself and adapt them to your own environment”. A design pattern is just that, “a model or guide for something to be made” (“Pattern,” 2009). Patterns can be useful in teaching other, less experienced, developers and also develop a standard vocabulary so everyone can understand the overall design (Fowler, 2003, p. 57). The “Model-View-Controller (MVC) is a widely used software design pattern ...[and] is a useful addition to a toolkit, no matter what language you choose” (Kotek, 2002, ¶ 2). MVC is a logical separation between the View; the user interaction piece, the Model; the business rules and data processing piece and the Controller; the communication avenue between the Model and View (Kotek, 2002). The most important aspect of MVC is that because of the disconnected nature, the different pieces can be changed without affecting the other pieces.

Object-Oriented (OO) Development concepts are essential in designing and developing web applications. Armstrong (2006) highlights the need to understand OO development concepts:

Understanding what concepts characterize OO is of paramount importance to both practitioners in the midst of transitioning to the OO approach and researchers studying the transition to OO development. How can we hope to achieve the productivity gains promised by the OO development approach, effectively transition software developers, or conduct meaningful research toward these goals, when we have yet to identify and understand the basic phenomena? (p. 124)

Armstrong (2006) goes on to state that “an established set of fundamental OO concepts within a taxonomy may enhance the maturity of the OO development discipline through standardization, and increase the portability of developers across organizations and environments”. A firm understanding of OO concepts implemented in an application can assist a designer in maximizing the scalability, extensibility and usability of the application.

Davidson (2007) states that “the database is the cornerstone of pretty much every business project” (§ 8), data is the key, entering, storing, processing, manipulating and displaying data for whatever purpose the user wishes. The concept of an evolutionary database design discussed by Fowler and Sadalage (2003) has some merits, the “design of the system has to evolve through the various iterations of the software” (§ 2). Through tightly controlled change, the database is allowed to grow and mature throughout the life-cycle of the project (Fowler & Sadalage, 2003). Automated refactoring is very important in this type of development, everything is controlled; the database schema and test data is rebuilt to ensure integrity of the entire system (Fowler & Sadalage, 2003).

Performance is an important part of any database, the better a developer understands that, the better the application. Chen, Goes, Gupta and Marsden (2004) explored query patterns and found it is not possible “to find a single database structure that is best under all conditions[, but it is possible]... to identify database structures that perform robustly”. It is a worthwhile effort to identify and design for the most robust structure possible.

Fraternali (1999) explains about how data-intensive web applications will cope with the special requirements:

As has happened in the past with other emerging technologies such as databases and object-oriented programming languages, methodologies and software tools may greatly

help in mastering the complexity of innovative applications by fostering a correct understanding and use of a new development paradigm, providing productivity advantages, and thus reducing the risk inherent in application development and migration. (p. 228)

These are new types of “hybrid” applications, web application and information system (Fraternali, 1999).

Chapter 3 – Methodology

The research methodology will be that of design science research, building an example ASP.NET web application driven by a Microsoft SQL Server 2005 database to gain the knowledge of what are the best practices associated with designing and building those types of applications.

The example application will be a simple data in / data out type of application with basic functionality to search for records, select records, add, change and delete records. It will be constructed as a three-tier architecture, using the Model-View-Controller (MVC) architectural pattern taking full advantage of Object-Oriented (OO) development concepts. All of the CRUD, Create, Read, Update, and Delete operations will be handled by Stored Procedures on the database side. Data change logging will be handled by Insert, Update and Delete triggers on the data table, inserting values into a specific log table. Application security will be handled by Active Directory (AD) for domain access and the system users, along with their associated security levels stored in database tables, used by the web user interface to control what a particular logged in user can do or see.

Chapter 4 – Project Analysis and Results

The example application was designed to replace an existing application that was cumbersome and time consuming to use, as well as, lacking key functionality and scalability required by the users to perform their job. The users requested an application that was simpler and more streamlined; facilitating quick, easy and reliable user interaction. In addition to the user experience, they also requested enhanced security features as well as increased flexible search features.

The new Audit Action Tracker (AAT) application will make the user interface simpler for the end user by taking advantage of Web-based technology. AAT redesigns the storage of data in a Microsoft SQL Server 2005 database using tables, triggers, procedures and functions to better manage data and develop reports with Crystal Reports XI for display in a company-wide reporting system, BusinessObjects Enterprise XI. This system will meet the functional and security requirements by managing the data, capturing an audit trail, and making the data more accessible and reportable.

Initial Design

Initial design is the cornerstone of any good application, the better the foundation the better the implementation and the more that the users will accept it. The more questions that can be answered in the beginning, the smoother the actual development will be. In small Information Technology (IT) organizations, the developer has to perform all the different jobs in the application development life-cycle, becoming the architect, the designer, the coder, the documenter, the tester as well as the developer. With all the responsibility falling on the

developer to do everything, it is important for the developer to address three areas in the initial design phase, user interaction, requirements gathering and documentation.

User Interaction

With regard to a business application, the user is the most important element. The business application itself is a tool that helps the user do their work more easily and efficiently. The user is a wealth of knowledge in what will help them perform their work better. The users know what they want or do not want in an application, so it is imperative to involve them in every aspect of the design and development process to ensure they will be satisfied with the finished product. Hager, Kibler and Zach (1999) said it best:

Besides the challenges imposed by web technology, the real challenges of web applications involve getting the content right so the users can do the work they need to and leave satisfied. The only way to get it right is with early and continuous focus on users and their tasks. Without user-centered design, applications are almost always frustrating to use, forcing many users to leave without doing what they came to do.

Obviously this isn't good for a company's bottom line. (p. 59)

If users feel they have a voice in designing the application, they will take more pride in working on it, work harder understanding what they truly need and when it is all finished, will take ownership in using the application.

Start small with a core group of users, the most knowledgeable and experienced, and then expand the group to involve more of the user community to get a wider perspective. This will allow the developer to have a basic understanding and be able to present ideas for feedback from the user group.

Weekly meetings are crucial to the success of the process, providing a forum for users to see the progress and give feedback. Having an agenda and sticking to it, is a good structure to help in the design meeting process. Meetings are not value-added if they are constantly off track, other unrelated topics are introduced or the users are uninterested or otherwise occupied.

Getting the users involved in testing the application or validating a prototype of the application will give them a better understanding of what they like and do not like and what the application can do. If the users see it in operation and work with it, they will provide better feedback to the developer. In most cases, for small IT organizations, it is not practical to build a prototype, but if pieces of the application, the most representative functionality pieces, can be built, the users can extrapolate what the other pieces of the application will be like.

Requirements Gathering

Understanding how an application is to be used, who is to use it and what they will be doing with it is very important for the developer. The developer must have an understanding of the overall process and the high-level functionality required for a user to perform their work activities and, once those are understood, they must be able to compile and articulate that information back to the users in a logical, understandable format, so that the user understands how the application will look and perform. Developers must understand and determine the source of requirements to build the application use case diagrams and data elements.

Source of requirements.

There are two main sources of requirements; replacing an existing application and user developed requirements. . Based on the different sources of requirements, are different approaches the developer must take to extract the information needed to formulate the initial

Table 1: An analysis of the pros and cons of the different sources of requirements. The table below contains data on the pros and cons of replacing an existing application vs. user requirements.

Source	Pros	Cons
Replacing an Existing Application	<ul style="list-style-type: none"> • Users will have a better understanding of what they like and do not like in an application • Developers will be able to work with the existing application to gain a better understanding of the user requirements • Both users and developers will have a common frame of reference 	<ul style="list-style-type: none"> • Users may be fixated on how the existing application works and not receptive to process changes... "this is the way we have always done it" • Users may feel threatened by a replacement application in either workload or employment • The application itself may be too complex to replicate in whole or part
User Developed Requirements	<ul style="list-style-type: none"> • The application is a blank slate, specific process and work-flow can be built exactly to the user specifications • Users will be focused on functionality and process and will be free to think outside what they already know and help streamline their work • Developers will have the opportunity to interject new and simpler processes into the design that the user may not have considered 	<ul style="list-style-type: none"> • The application is a blank slate, if the user requirements are too vague, the process to understand and refine the design may be a long process • Developers have to work harder to understand functionality and process, more interaction with the users and more example development must be done • Based on arbitrary, grandiose requirements and development limitations the application itself may be too complex to build

design. When the developer is replacing an existing application, the focus is on how the existing application works and how the users want to add to or change the functionality to better suit their needs. The developer must be granted access to the existing application in a test environment and given user contacts to work help understand the required functionality. As the developer is working with user identified requirements, the focus is on understanding the vision of the users, what functionality they require, want and would like to have. The developer must work closely with the users to constantly refine the ideas into a workable design.

There are advantages and disadvantages to both sources of requirements as shown in Table 1. There are also different approaches to gathering requirements for both sources of requirements. The developer must understand these to be able to successfully develop an initial design. Once the developer has the underlying understanding of the requirements, the building of the artifacts begins.

Defining use cases.

The use case is the focal point of the initial design, it is the definition of the specific activity to be performed and which specific actors will be performing them. The use cases can be presented in two forms, the use case diagram shown in Figure 1 and the use case narrative shown in Figure 2.

The use case diagram and the use case narrative most often go together when detailing the use case design. The use case diagram is a good visual representation of the interaction between the actor and an application function. The use case narrative is a textual representation of the interaction between the actor and an application function with more detail.

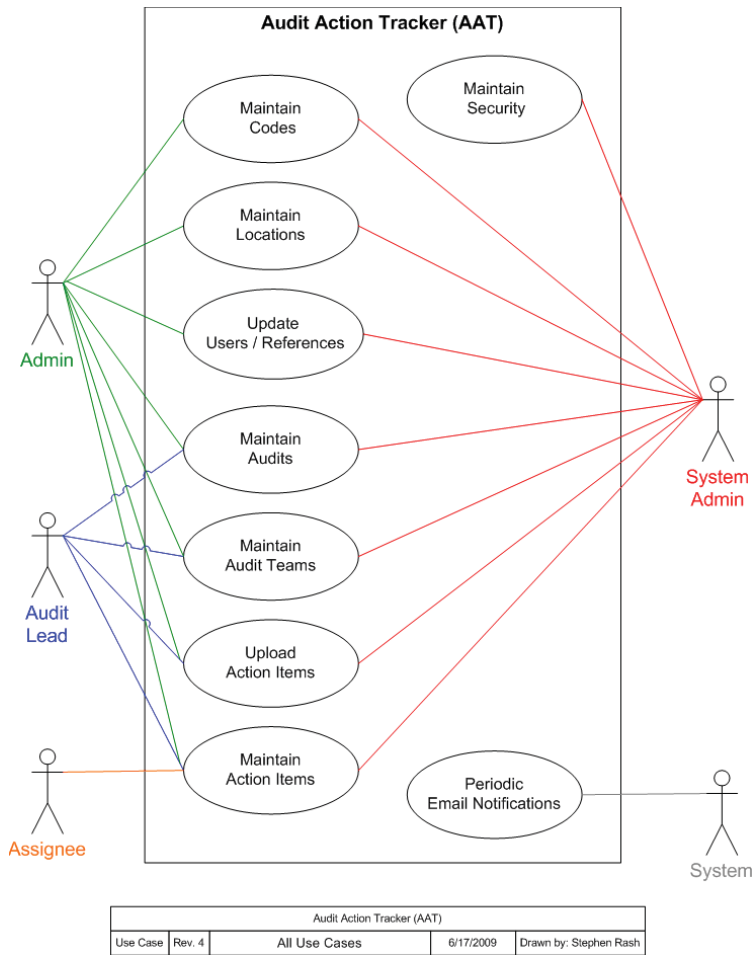


Figure 1: Use Case Diagram.

Name	Maintain Codes
Number	1.0
Description	Allow the user to maintain the code lists that populate drop-down lists in the application.
Author	Stephen C. Rash
Date	7/7/2009
Actors	Admin and System Admin
Pre-conditions	<ol style="list-style-type: none"> 1. Database connection established. 2. User possesses proper security to use case and functions 3. Code Type must already be in the system
Actions	<ol style="list-style-type: none"> 1. Search for Codes 2. Select Code Record 3. Add Code Records 4. Change Code Records 5. Delete Code Records
Post-conditions	<ol style="list-style-type: none"> 1. Changes to Code Record will be logged
Includes	None
Extends	None
Generalizes	None

Figure 2: Use Case Narrative.

Defining data elements.

Once the use cases are defined, the developer then must delve into more detail about what specific data elements must be captured. The data elements drive the database design as well as the actual model, the page and user control design in the application. As shown in Figure 3, a simple spreadsheet can contain all the pertinent information the developer needs to do the actual implementation.

Label Name	Field Name	DB Type	Size	Nullable	Primary Key	Foreign Key	IX (Unique)	SP Modify	SP Select	SP Lookup
<Hidden>	code_id	INT IDENTITY		No	Yes			Yes	Yes	Yes
Code Type	code_type_id	INT		No		tbl_code_type(code_type_id)	Yes	Yes	Yes	Yes
Code	code_name	VARCHAR	50	No			Yes	Yes	Yes	Yes
Code Dependency	code_dep_id	INT		Yes		tbl_code(code_id)		Yes	Yes	Yes
Active	code_active	BIT		No				Yes	Yes	Yes
<Hidden>	code_lmod_user	VARCHAR	20	No				Yes	Yes	
<Hidden>	code_lmod_date	DATETIME		No				Yes	Yes	
<Hidden>	code_lmod_type	CHAR	1	No				Yes	Yes	

Figure 3: Data Elements Spreadsheet.

The data element lists will allow the developer to map out the construction of the database tables, primary and foreign keys, unique index, stored procedures and how they relate to each other, as well as the application domain objects, implementations and the pages/controls themselves. It is an important tool to keep updated as a reference to what the pieces should all look like.

Defining visual aids.

A picture really is worth a thousand words, but only if it is understandable to the consumer. It is important to present the different functions to the user in a visual form, to show what the look and feel of the application will be. The user will be able to take that visual aid and be able to comment and make changes to the overall design of the functions.

Figure 4 is a visual example of a specific use case and how it might be built in the application; the user can see the navigation, see the different functions available and see how the data elements will be incorporated into the application.

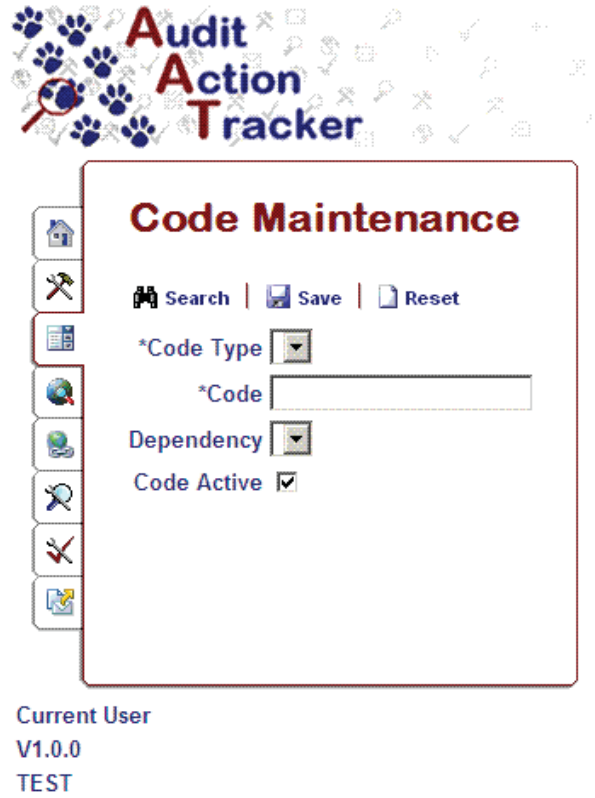


Figure 4: Use Case Visual Aid.

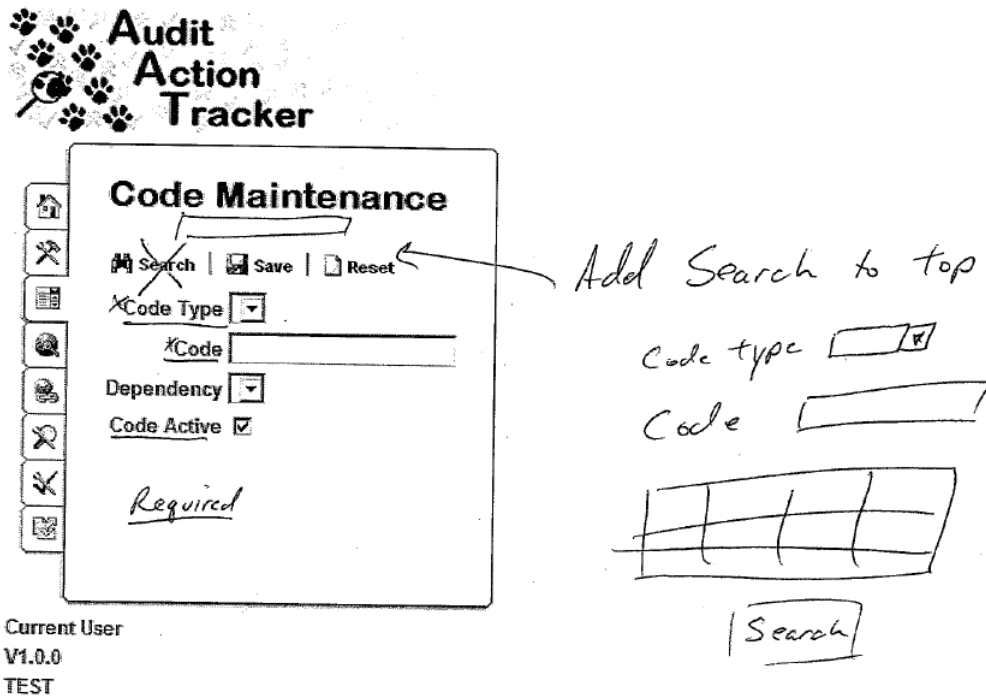


Figure 5: Marked-up Use Case Visual Aid.

Figure 5 shows the visual aid after it has been marked-up by the users. When the look and feel of the application has been refined and finalized the developer can start work on a prototype or example function of the application for the users to see. More revisions may be necessary, but with a little extra time in the initial design, coding revisions will more likely be minimal.

Documentation

Documentation is very important to a successful project as it allows all the individuals involved to take time to think out the aspects of the application and write it down so that everyone has a source for understanding what is required, how it will be produced and that the responsible individuals approve.

Producing the scope document.

The scope document is a detailed document that defines the justification for “why” the application is being developed, the high-level usage scenarios, what the application will exclusively entail and specifically, what it will not. In smaller organizations, this document is the collaboration between the users and the developers to produce and refine it into something that all can agree upon.

Producing the design document.

The design document is a compilation of the finalized design artifacts produced by the developer. The main purpose of this document is for the developer to have something to work from and also, so any future projects or developers can see how the application was designed and take away that knowledge.

User design acceptance and signoff.

The acceptance and signoff document is user approval to move forward. This could consist of a single page signed by the user or an email stating the user has reviewed and approved the initial design and the development work can proceed.

Security Design

The initial design is the blueprint of the application; the security is the gated fence around the application through which no one passes without authorization. Security is a very important aspect to any business application large or small, windows or web, you must control who has access and at what level. The developer must take into consideration many aspects of security, how many types of individuals will access the application, what level of granularity the security must be and how the security will be presented to the users. The level of complexity depends upon the application, from any user having access to the application with access to all functions to a particular user having access to only particular functions. Internal business applications have traditionally encompassed all aspects of application security; they stored the user's passwords and access levels. More recently, internal business applications have the advantage of being able to take benefit of network security to control application access. The developer has to build on the network security to store user and access information in the database to be used to specify access within the application. There are advantages and disadvantages to both of these security architectures. Table 2 shows the advantages and disadvantages to both of these security architectures.

Network Setup

All users that will access the application must have network security credentials. When the user accesses the network, they automatically have access to the application without having

Table 2: An analysis of the pros and cons of the different security architectures. The table below contains data on the pros and cons of a traditional application based security vs. network enabled security.

Security Architecture	Pros	Cons
Traditional Application Based Security	<ul style="list-style-type: none"> • Better access control to the application, all user ids and passwords are stored in the database and the application is more secure • System administrators would be able to more quickly grant users access to the application • The user does not need to have network access to be setup in the application 	<ul style="list-style-type: none"> • More maintenance issues because all user's passwords are stored in the application and must be encrypted • Logon is required each time the user accesses the application • The user would have to remember a different password to access the application
Network Enabled Security	<ul style="list-style-type: none"> • Seamless security into the application; the user is authenticated by network security and proceeds into the application • If a user's network security access is disabled, they would be disabled in the application at the same time • Easier user maintenance, ties the user's network id to what security access they require, no password storage 	<ul style="list-style-type: none"> • There is a security risk, if a user does not lock their computer when not in the area, so anyone could access the application AS that particular user from that computer • The user is required to have network security credentials to access the application • More than a single individual must be involved in setting up application access

to login again. Seamless access and not having to remember another password is a big advantage to the users. The security risk to this type of architecture can be mitigated by password protected screen savers and user training. The advantages outweigh the risks.

Database Setup

The database will contain at least one table that contains the user's network identification and their security access. More complex security architecture will contain two or more tables to hold users, groups and security information.

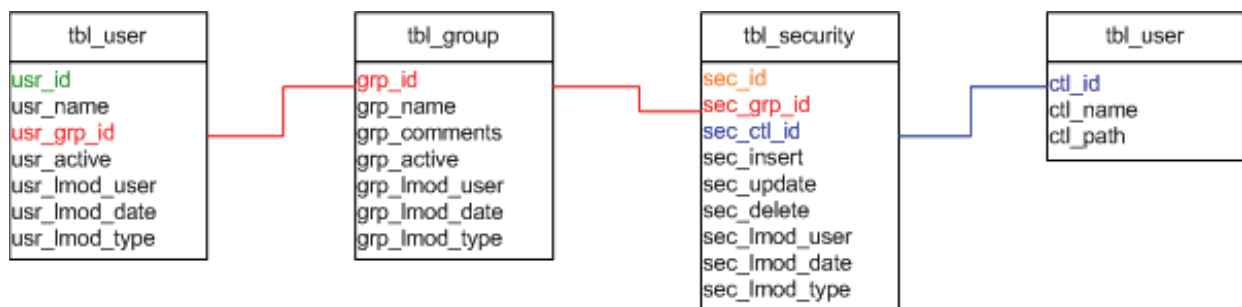


Figure 6: Multiple Table Security Setup.

The database will also contain a stored procedure that accepts parameters for user and control to return the level of security access.

Application Setup

The application will access the user – security information via a stored procedure in the database to grant access to the particular page and or user control with its specific insert, change, delete and search functions. Figure 7 shows the flow of the application security. The application security is based on the accessed page or control and the network user id. The page or user control and user id is passed to the database and a security record is returned consisting of the level of access that user has to that particular control, the including ability of the user to insert, update or delete records.

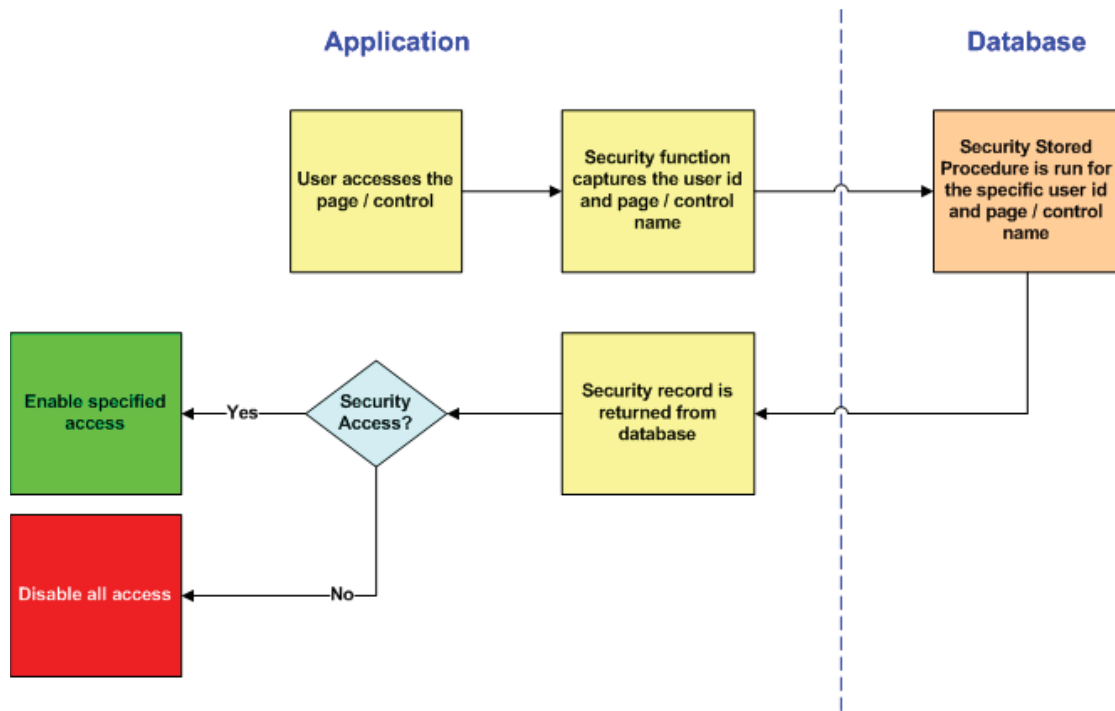


Figure 7: Application Security Setup.

Database Design

As the initial design is the blueprint of the application, the security is the fence around the application, so then it follows that the database is the foundation of the application. The database development must be the first step in actually building the application. The database has two main functions, store the data and facilitate the CRUD, Create, Read, Update and Delete operations.

Database Users

The Achilles heel of security for a database driven web-based application is the double-hop authentication, client to web server to database server. A simple solution to this issue is to create database user application id, which has full select, insert, update and delete access to the tables, as well as execute access to all stored procedures. The application id is used in the application to connect to the database and perform the operations. Another handy database user

is a read-only id, which has only select access to the tables and also has execute access to selected stored procedures. The read-only id is used in reporting and any external access to the application data.

Tables

In a database, the table is the most important piece; it is where the data is stored, the particular data in the table must be unique, consistent and retrievable. The best data table architecture is as follows:

- A next sequential record key, an integer identity field which is the tables primary key
- Data type specific fields, the correct data type for the type of data it will be accepting, every unused bite of data will still take up database space
- Fields accepting null values sparingly, if the field may not have any data populated, the field should accept null values, but if the field will more likely have values, the field should not accept null values.
- If the record may be changed by different users, logging should be incorporated into the table design
- For data integrity, foreign key constraints should be placed on fields where the values relate to the primary key of other tables
- At least, a unique index should be used, the literal name or unique values captured in a table
- Grant full select, insert, update and delete permissions to the application user and grant select only permissions to the read-only user

Figure 8 shows the example SQL code for a data table.

```

USE AAT
GO
/* Drop Table */
IF EXISTS (
    SELECT *
    FROM sys.objects
    WHERE name = 'tbl_code'
    AND type = 'U'
)
DROP TABLE dbo.tbl_code
GO
/*****
* DATABASE: AAT
* TABLE: tbl_code
* CREATED BY: Stephen C. Rash
* CREATED DATE: 07/14/2009
* UPDATED BY: Stephen C. Rash
* UPDATED DATE: 07/20/2009
* PURPOSE: Holds code data records.
*****/
/* Build Data Table */
CREATE TABLE dbo.tbl_code (
    code_id INT IDENTITY NOT NULL,
    code_type_id INT NOT NULL,
    code_name VARCHAR(50) NOT NULL,
    code_dep_id INT NULL,
    code_active BIT NOT NULL,
    code_lmod_user VARCHAR(20) NOT NULL,
    code_lmod_date DATETIME NOT NULL,
    code_lmod_type CHAR(1) NOT NULL,
    CONSTRAINT pk_code PRIMARY KEY CLUSTERED (code_id),
    CONSTRAINT fk_code_type_id FOREIGN KEY (code_type_id)
        REFERENCES tbl_code_type (code_type_id),
    CONSTRAINT fk_code_dep_id FOREIGN KEY (code_dep_id)
        REFERENCES tbl_code (code_id)
)
GO
/* Indexes */
CREATE UNIQUE INDEX ix_code_unique
ON dbo.tbl_code (code_type_id, code_name)
GO
/* Permissions */
GRANT SELECT, INSERT, UPDATE, DELETE
ON dbo.tbl_code
TO AATAPP
GO
GRANT SELECT
ON dbo.tbl_code
TO AATRO
GO

```

Figure 8: Data Table Structure.

Logging

Logging is an important feature to determine the “who,” “what” and “when” a table record was changed. If the table has records that could potentially be updated by different users, logging is required. The logging feature is simple and consists of a log table which is structured very similarly to the data table and a trigger that executes on a record change in the data table.

The log table has the exact same fields as the data table, the only differences are that the log table has no primary key or foreign keys and its unique index is the data table's primary key field and the logging fields. The example SQL code for a log table is shown in Figure 9.

```

USE AAT
GO
/* Drop Table */
IF EXISTS (
    SELECT *
    FROM sys.objects
    WHERE name = 'tbl_code_log'
    AND type = 'U'
)
DROP TABLE dbo.tbl_code_log
GO
/*****
* DATABASE: AAT
* TABLE: tbl_code_log
* CREATED BY: Stephen C. Rash
* CREATED DATE: 07/14/2009
* UPDATED BY: Stephen C. Rash
* UPDATED DATE: 07/20/2009
* PURPOSE: Holds code log records from tbl_code.
*          The who, what and when the record was inserted,
*          changed or deleted by use of a Trigger.
*****/
/* Build Log Table */
CREATE TABLE dbo.tbl_code_log (
    code_id INT NOT NULL,
    code_type_id INT NOT NULL,
    code_name VARCHAR(50) NOT NULL,
    code_dep_id INT NULL,
    code_active BIT NOT NULL,
    code_lmod_user VARCHAR(20) NOT NULL,
    code_lmod_date DATETIME NOT NULL,
    code_lmod_type CHAR(1) NOT NULL
)
GO
/* Indexes */
CREATE UNIQUE INDEX ix_code_unique
ON dbo.tbl_code_log (
    code_id,
    code_lmod_user,
    code_lmod_date,
    code_lmod_type
)
GO
/* Permissions */
GRANT SELECT, INSERT, UPDATE, DELETE
ON dbo.tbl_code_log
TO AATAPP
GO
GRANT SELECT
ON dbo.tbl_code_log
TO AATRO
GO

```

Data fields

Log fields

Unique index of Code record key and Code log fields

Permissions for application id

Permissions for read-only id

Figure 9: Log Table Structure.

The logging trigger is executed on any insert, update or delete and copies the data table record exactly and inserts it into the log table. Figure 10 shows the example SQL code for the log trigger.

```

USE AAT
GO
/* Drop Trigger */
IF EXISTS (
    SELECT *
    FROM sys.objects
    WHERE name = 'tgr_code_log'
    AND type = 'TR'
)
DROP TRIGGER dbo.tgr_code_log
GO
/* Build Trigger */
CREATE TRIGGER dbo.tgr_code_log
ON tbl_code
AFTER INSERT, UPDATE, DELETE
AS
/*****
* DATABASE: AAT
* TRIGGER: tgr_code_log
* TRIGGER ON: tbl_code
* CREATED BY: Stephen C. Rash
* CREATED DATE: 07/14/2009
* UPDATED BY: Stephen C. Rash
* UPDATED DATE: 07/15/2009
* PURPOSE: Logs events to the tbl_code.
*****/
/* Return Errors */
IF @@rowcount = 0 RETURN
IF @@error != 0 RETURN
/* Log Type */
IF EXISTS (select * from inserted)
    INSERT INTO tbl_code_log
    SELECT code_id = code_id,
           code_type_id = code_type_id,
           code_name = code_name,
           code_dep_id = code_dep_id,
           code_active = code_active,
           code_lmod_user = code_lmod_user,
           code_lmod_date = code_lmod_date,
           code_lmod_type = code_lmod_type
    FROM inserted
ELSE
    INSERT INTO tbl_code_log
    SELECT code_id = code_id,
           code_type_id = code_type_id,
           code_name = code_name,
           code_dep_id = code_dep_id,
           code_active = code_active,
           code_lmod_user = code_lmod_user,
           code_lmod_date = code_lmod_date,
           code_lmod_type = code_lmod_type
    FROM deleted
GO

```

Trigger on Code record insert, update or delete

If in the inserted table, record is a insert or change, if not, a delete

Log an exact copy of the inserted or changed Code record

Log an exact copy of the deleted Code record

Figure 10: Logging Trigger Structure.

The logging is simple but effective, the log table, based on the last modification date-time field in descending order, can show the administrator what changes have been made to the record throughout its life and what user made the change.

Stored Procedures

The table is the most important piece in a database, but the stored procedures are a very close second. The stored procedures control the record manipulation and retrieval of data from the data tables. There are three main types of stored procedures, lookup, select and modify.

```

USE AAT
GO
/* Drop SP */
IF EXISTS (
    SELECT *
    FROM sys.objects
    WHERE name = 'sp_code_lookup'
    AND type = 'P'
)
DROP PROCEDURE dbo.sp_code_lookup
GO
/* Build SP */
CREATE PROCEDURE dbo.sp_code_lookup (
    @code_type_id VARCHAR(9),
    @code_name VARCHAR(50),
    @code_type_dep_id VARCHAR(9),
    @code_dep_id VARCHAR(9),
    @code_active CHAR(1)
) WITH RECOMPILE
AS
/*****
* DATABASE: AAT
* PROCEDURE: sp_code_lookup
* CREATED BY: Stephen C. Rash
* CREATED DATE: 07/14/2009
* UPDATED BY: Stephen C. Rash
* UPDATED DATE: 08/13/2009
* PURPOSE: Lists code table record information.
*****/
/* Select Record */
SELECT TOP 201
    c.code_id,
    dbo.fn_code_type_name(c.code_type_id) AS code_type_name,
    c.code_name,
    ISNULL(dbo.fn_code_type_name(t.code_type_dep_id), '') AS
        code_type_dep_name,
    ISNULL(dbo.fn_code_name(c.code_dep_id), '') AS code_dep_name,
    dbo.fn_yes_no(c.code_active) AS code_active_name
FROM tbl_code c
JOIN tbl_code_type t
ON c.code_type_id = t.code_type_id
WHERE c.code_type_id LIKE @code_type_id
AND c.code_name LIKE @code_name
AND ISNULL(t.code_type_dep_id, 0) LIKE @code_type_dep_id
AND ISNULL(c.code_dep_id, 0) LIKE @code_dep_id
AND c.code_active LIKE @code_active;
GO
/* Permissions */
GRANT EXECUTE
ON dbo.sp_code_lookup
TO AATAPP, AATRO
GO

```

Accepts Code search criteria parameters

Returns Top 201 Code records

Based on the Code search criteria

Permissions for application and read-only ids

Figure 11: Lookup Stored Procedure.

The lookup stored procedure.

The lookup stored procedure accepts search criteria parameters to return a list of records to populate the search facility in the application. The search is limited to 200 records for application performance, if the returned record set is the max, 201, the user will be asked to limit the search. Conversely, if there are no records returned for the search criteria, the user will be advised of that as well. The SQL code shown in Figure 11 is an example of a lookup stored procedure.

```

USE AAT
GO
/* Drop SP */
IF EXISTS (
    SELECT *
    FROM sys.objects
    WHERE name = 'sp_code_select'
    AND type = 'P'
)
DROP PROCEDURE dbo.sp_code_select
GO
/* Build SP */
CREATE PROCEDURE dbo.sp_code_select (
    @code_id INT
) WITH RECOMPILE
AS
/*****
 * DATABASE: AAT
 * PROCEDURE: sp_code_select
 * CREATED BY: Stephen C. Rash
 * CREATED DATE: 07/14/2009
 * UPDATED BY: Stephen C. Rash
 * UPDATED DATE: 07/20/2009
 * PURPOSE: Selects code table record information.
 *****/
/* Select Record */
SELECT code_id,
       code_type_id,
       code_name,
       ISNULL(code_dep_id,0) AS code_dep_id,
       code_active,
       code_lmod_user,
       code_lmod_date,
       code_lmod_type
FROM tbl_code
WHERE code_id = @code_id;
GO
/* Permissions */
GRANT EXECUTE
ON dbo.sp_code_select
TO AATAPP
GO

```

Accepts a single parameter,
the Code record key

Return a single Code record

Based on the Code record key

Permissions for application id

Figure 12: Select Stored Procedure.

```

USE AAT
GO
/* Drop SP */
IF EXISTS (
    SELECT *
    FROM sys.objects
    WHERE name = 'sp_code_modify'
    AND type = 'P'
)
DROP PROCEDURE dbo.sp_code_modify
GO
/* Build SP */
CREATE PROCEDURE dbo.sp_code_modify (
    @code_id INT,
    @code_type_id INT,
    @code_name VARCHAR(50),
    @code_dep_id INT,
    @code_active BIT,
    @code_lmod_user VARCHAR(20),
    @code_lmod_date DATETIME,
    @code_lmod_type CHAR(1)
) WITH RECOMPILE
AS
/******
* DATABASE: AAT
* PROCEDURE: sp_code_modify
* CREATED BY: Stephen C. Rash
* CREATED DATE: 07/14/2009
* UPDATED BY: Stephen C. Rash
* UPDATED DATE: 07/15/2009
* PURPOSE: Modifies code table record information.
*****/
/* Declarations */
DECLARE @hold AS TABLE (hold_id INT);
DECLARE @ret_id AS INT;
IF @code_lmod_type = 'D' ----- If 'D' - Delete
BEGIN
    /* Disable Log Trigger */
    DISABLE TRIGGER tgr_code_log ----- Disable Code Log trigger
    ON tbl_code;
    /* Update Last Mod */
    UPDATE tbl_code
    SET code_lmod_user = @code_lmod_user,
        code_lmod_date = @code_lmod_date,
        code_lmod_type = @code_lmod_type ----- Update Code log fields
    WHERE code_id = @code_id; ----- Based on the Code record key
    /* Enable Log Trigger */
    ENABLE TRIGGER tgr_code_log ----- Enable Code Log trigger
    ON tbl_code;
    /* Delete Record */
    DELETE ----- Delete Code record
    FROM tbl_code ----- Based on the Code record key
    WHERE code_id = @code_id; ----- Based on the Code record key
    /* Set Return Value */
    SET @ret_id = @code_id; ----- Set Deleted Code key
END
ELSE IF @code_id = 0 ----- If code_id = 0 (New)
BEGIN
    /* Insert Record */
    INSERT INTO tbl_code
    OUTPUT inserted.code_id INTO @hold
    SELECT code_type_id = @code_type_id,
        code_name = @code_name,
        code_dep_id = @code_dep_id,
        code_active = @code_active,
        code_lmod_user = @code_lmod_user,
        code_lmod_date = @code_lmod_date,
        code_lmod_type = @code_lmod_type; ----- Insert Code record and
    /* Set Return Value */
    SELECT @ret_id = hold_id FROM @hold ----- Set Deleted Code key
END
ELSE
BEGIN
    /* Update Record */
    UPDATE tbl_code
    SET code_type_id = @code_type_id,
        code_name = @code_name,
        code_dep_id = @code_dep_id,
        code_active = @code_active,
        code_lmod_user = @code_lmod_user,
        code_lmod_date = @code_lmod_date,
        code_lmod_type = @code_lmod_type ----- Update Code record
    WHERE code_id = @code_id; ----- Based on the Code record key
    /* Set Return Value */
    SET @ret_id = @code_id; ----- Set Deleted Code key
END
/* Select ID Out */
SELECT @ret_id; ----- Return Code key
RETURN;
GO
/* Permissions */
GRANT EXECUTE ----- Permissions for application id
ON dbo.sp_code_modify
TO AATAPP
GO

```

Figure 13: Modify Stored Procedure.

The select stored procedure.

The select stored procedure accepts a single parameter, the record key, to return a single record to populate the data manipulation facility in the application. An example of the SQL code to build the select stored procedure is shown in Figure 12.

The modify stored procedure.

The modify stored procedure accepts all record parameters and based on the type of modification to be performed, manipulates the record in the table and then returns the record key for the messaging facility in the application. Figure 13 shows the example SQL code for a modify stored procedure.

```

USE AAT
GO
/* Drop Function */
IF EXISTS (
    SELECT *
    FROM sys.objects
    WHERE name = 'fn_code_name'
    AND type = 'FN'
)
DROP FUNCTION dbo.fn_code_name
GO
/* Build Function */
CREATE FUNCTION dbo.fn_code_name (
    @code_id INT
) RETURNS VARCHAR(50)
AS
/*****
* DATABASE: AAT
* FUNCTION: fn_code_name
* CREATED BY: Stephen C. Rash
* CREATED DATE: 07/14/2009
* UPDATED BY: Stephen C. Rash
* UPDATED DATE: 08/25/2009
* PURPOSE: To retrieve code_name value based on id.
*****/
BEGIN
    /* Return Value */
    DECLARE @code_name VARCHAR(50);
    SET @code_name = '';
    SELECT @code_name = code_name
    FROM tbl_code
    WHERE code_id = @code_id;
    RETURN @code_name;
END
GO
/* Permissions */
GRANT EXECUTE
ON dbo.fn_code_name
TO AATAPP, AATRO
GO

```

Accepts a single parameter,
the Code record key

Select Code name value
Based on the Code record key
Return Code name

Permissions for application and read-only ids

Figure 14: Scalar Function.

Functions

The scalar function accepts a single parameter and returns a single descriptive value. The scalar function is used to decode specific values in the database. The example SQL code for a scalar function is shown in Figure 14.

The Code Table

Most applications have lists of values that have meaning and populate data records. Storing of these lists in an application can be architected in two ways: a more traditional architecture where there are several tables each correlating to a specific list type or a single table using a type to store all lists. The code table concept is the storing of all lists in a single code table with an associated code type to differentiate each list and a hierarchical structure to identify dependencies for use in cascading drop-down lists in the application. There are advantages and disadvantages to both types of architecture. Table 3 shows the advantages and disadvantages to both types of architecture.

Which list architecture is the best? That depends on which advantages will best serve the application and which disadvantages will be the least detrimental. A combination, a hybrid, of both of these architectures is going to be the best, then the static constrained lists can be incorporated into the code table and the more complex lists can be separated into different tables.

Application Design

With the initial design being the blueprint, security being the fence, the database being the foundation, then the user interface becomes the actual building with a well thought out floor plan, easy for everyone to use and beautiful to look at. Once the database development is complete, the next step is to build the application. Just like in a construction project, the development must be from the ground up, it is difficult to start on the fourth floor and work

Table 3: An analysis of the pros and cons of the different list storage architectures. The table below contains data on the pros and cons of a traditional separate table vs. the code table architecture.

List Architecture	Pros	Cons
Traditional Separate Table Architecture	<ul style="list-style-type: none"> • The developer has more latitude to add additional data fields and use different data types • With separate tables, querying each list will be based on the number of records in each table, not all lists • Logical table names will allow for quick identification in the database 	<ul style="list-style-type: none"> • The application will be more complex, multiple points of entry, one for each list to be maintained • More development time will be required in the database to build the functions to search, select and update each list table • The application code to drive the drop-down lists for each list would be more complicated and each would have to be built individually
The Code Table Architecture	<ul style="list-style-type: none"> • The application will be less complex, the user would have a single point of entry to maintain all codes • Less development time will be required in the database, once the functions to search, select and update a single type of list are created, those functions can be reused in the application • The application code to drive the drop-down list can more easily be reused in the application 	<ul style="list-style-type: none"> • The developer is constrained to just an id and description, no other additional information can be captured unless it is captured for all codes • With a single table to hold all codes, the more records, the slower the querying capabilities of the application • The user would need to decode the record in the code table with the code type to identify the members of each list

down. In application design, the floors equate to the layers of the Layered (n-tier) Architecture: the ground floor is the Domain Layer, then the Service Layer, next the Business Layer and lastly, the Presentation Layer.

Layered (n-tier) Architecture

The Layered Architecture is a logical separation of high-level functionality. This type of architecture lends itself to being maintainable and distributable. It is maintainable because of the grouping of the similar functions and it is distributable because the different layers can be run on different physical hardware.

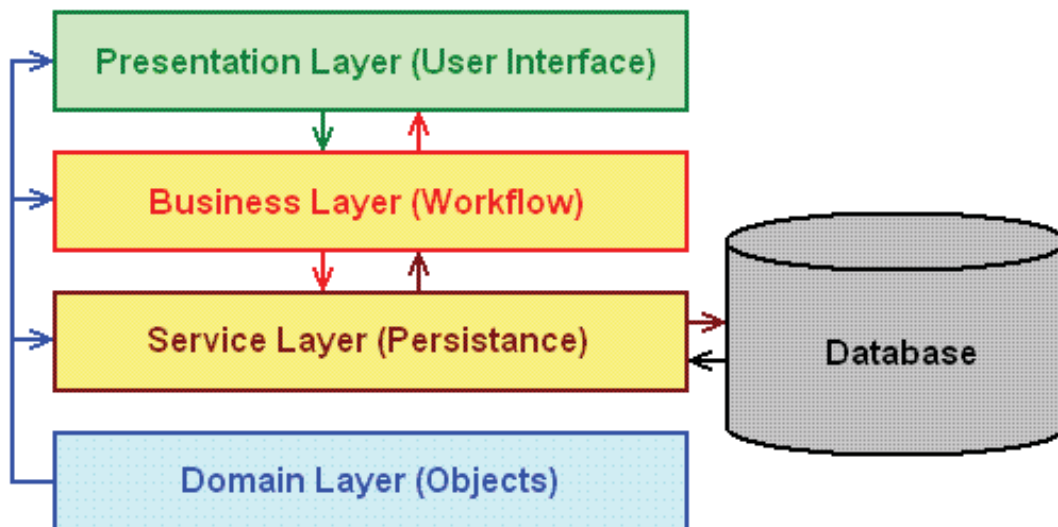


Figure 15: Layered Architecture.

Domain layer.

The domain layer contains the actual data classes, or objects, that the different layers use to move records within the application. The domain layer objects are use in all other layers of the application. The domain layer structure is shown in Figure 16.

Service layer.

The service layer contains the classes that handle the persistence, the movement of domain objects to and from the data store. The service layer also hides the specific persistence

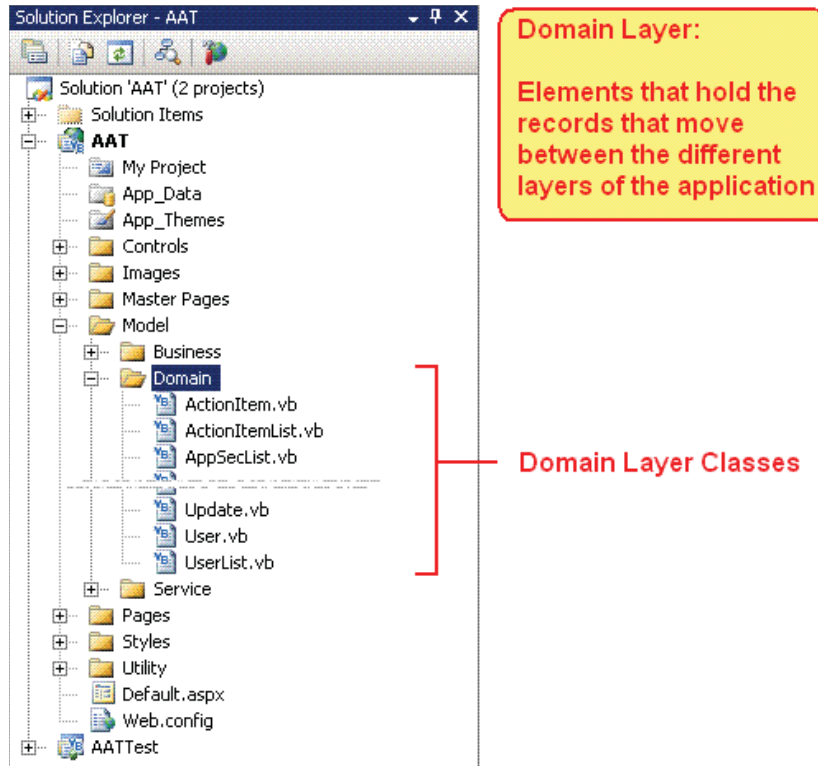


Figure 16: Domain Layer Structure.

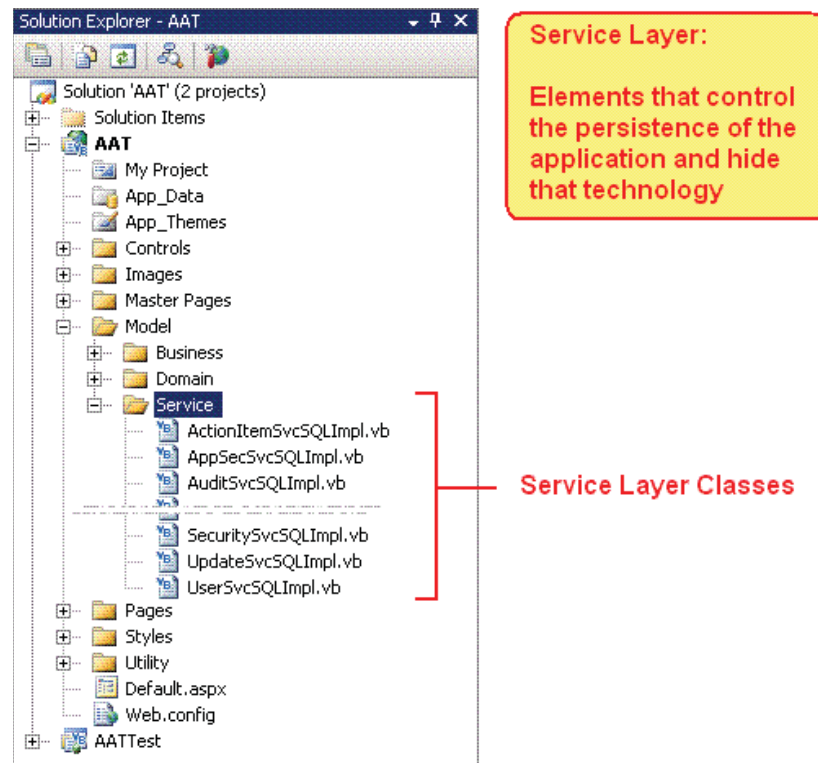


Figure 17: Service Layer Structure.

technology from the business layer, so it can be swapped out with another technology without affecting the business layer. Figure 17 shows the service layer structure.

Business layer.

The business layer contains the classes that handle the use case workflow, the specific rules under which the application operates. The business layer is also the main interface point to the presentation layer. The business layer structure is shown in Figure 18.

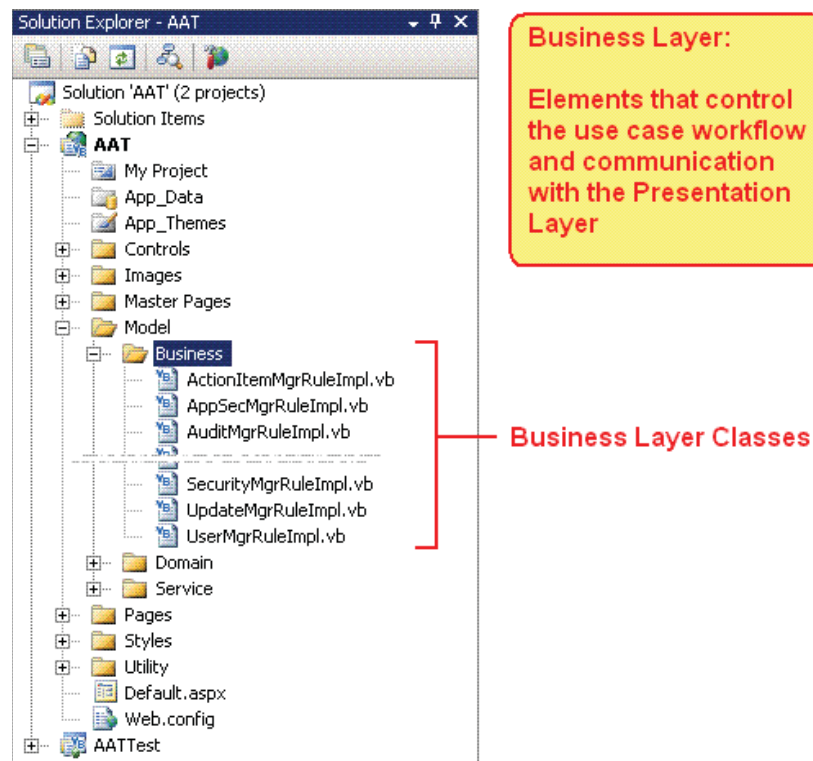


Figure 18: Business Layer Structure.

Presentation layer.

The presentation layer contains all the elements that handle the interaction with the user, known as the User Interface. The presentation layer displays the visual representation of the application to the user, accepts inputs and passes those inputs on to the business layer for processing, then, accepts return messages that the process was either successful or unsuccessful.

Figure 19 shows the presentation layer structure.

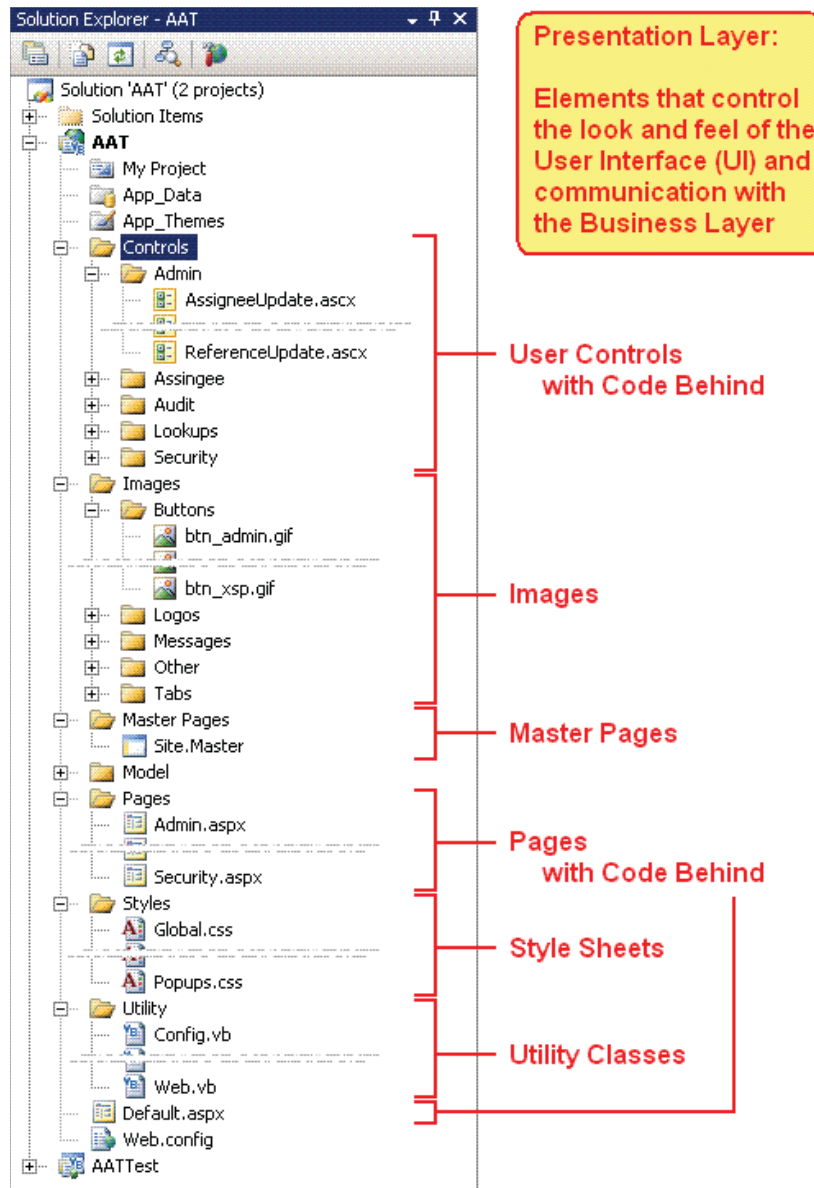


Figure 19: Presentation Layer Structure.

Design Patterns

Design patterns are a structured approach to designing elements of an application to take advantage of the principals of Object Oriented Programming (OOP). Inheritance, encapsulation and reuse, just to name a few, even the object and classes themselves are rooted in OOP and used in design patterns. Table 4 shows the different types of design patterns, how they are used and where they are used in the application.

Table 4: An analysis of the different design patterns. The table below contains data on how design patterns are used and where in the Layered Architecture they are used.

Design Pattern	What it is used for...	Where it is used...
Model, View, Controller (MVC)	<ul style="list-style-type: none"> • An overall architectural design pattern • To simplify the communication between the different elements • To separate like functions: <ul style="list-style-type: none"> ○ Model – the application logic, workflow, persistence and objects ○ View – the User Interface, the graphical representation of the application ○ Controller – the communication between the View and the Model 	<ul style="list-style-type: none"> • Presentation Layer • Business Layer • Service Layer • Domain Layer
Layer Supertype	<ul style="list-style-type: none"> • To dynamically instantiate the rules implementations by use of the Web.config file based on the requested manager interface • To provide a common interface between the Business Layer and Service Layer using the Factory 	<ul style="list-style-type: none"> • Business Layer
Separated Interface	<ul style="list-style-type: none"> • To decouple the higher level Manager or Service Interface from the actual implementation logic 	<ul style="list-style-type: none"> • Business Layer • Service Layer
Plugin	<ul style="list-style-type: none"> • To encapsulate the interface details and the implementations • To easily swap out the business rules in the Business Layer or persistence mechanism in the Service Layer 	<ul style="list-style-type: none"> • Business Layer • Service Layer
Marker (Serializable) Interface	<ul style="list-style-type: none"> • To provide a common interface to the Service Interfaces and Manager Interfaces 	<ul style="list-style-type: none"> • Business Layer • Service Layer

Factory	<ul style="list-style-type: none"> To dynamically instantiate the persistence implementations by use of the Web.config file based on the requested service interface 	<ul style="list-style-type: none"> Service Layer
Singleton	<ul style="list-style-type: none"> To ensure that only one object is instantiated in the Factory 	<ul style="list-style-type: none"> Service Layer
Object	<ul style="list-style-type: none"> To hold the actual data record 	<ul style="list-style-type: none"> Domain Layer

Figure 20 shows how the different design patterns can be employed throughout the application, to achieve the desired goal of a simple, maintainable and useable application.

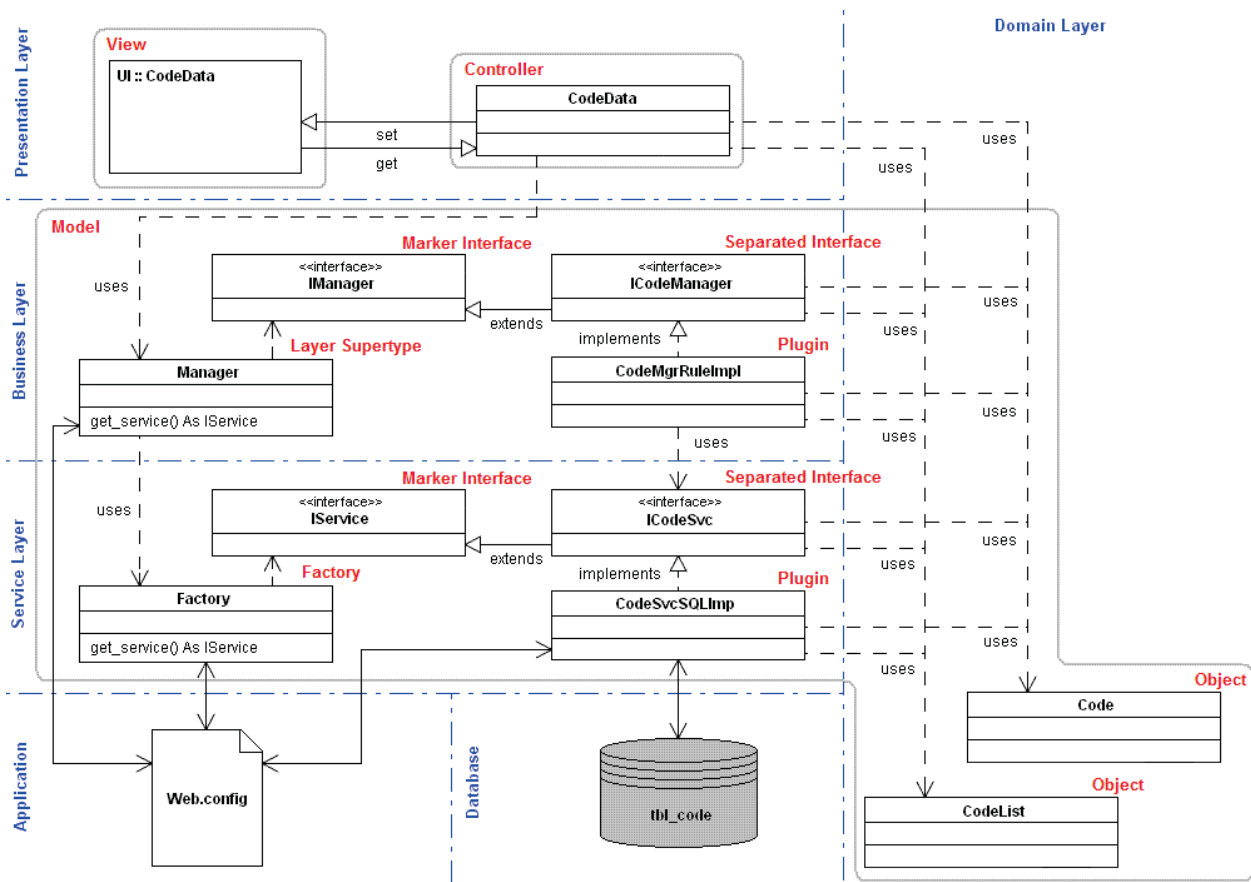


Figure 20: Application Design Patterns.

Design patterns assist the developer in standardizing the application; each function will be built in the same manner. Any other developer working on the application will be able to follow the methodology and easily integrate additional functions.

The Model

The Model is the part of the application that contains the workflow logic and persistence of the application. The Model consists of the classes that make up the Domain Layer, Service Layer and Business Layer of the application.

Domain Classes.

Domain classes are very important in Object-Oriented Programming, as they are the objects themselves. These classes are used throughout the application to move data around as a consistent record set. The Domain class is based on the table layout in the database; it consists of the same fields with the same high-level data types, so the first step in creating the object is to define the fields. Second, the object properties, the communication in and out of the object, must be defined through gets and sets. Next, the constructors are defined, a default constructor so the object can be instantiated without being populated and the overloaded constructor which accepts the values and populates the object. An override to string function is nice to have so the contents of the object can be viewed in a string format, but it is not necessary. A validation function is also nice to have to ensure the object is populated correctly, but again not necessary. An example of an example domain class is shown in Figure 21.

Service Classes.

Service classes handle the communication to the data store. The architecture includes a Factory class, an IService class, a service interface for each data function and, at least one service implantation for each data function.

The Factory class shown in Figure 22 is used to dynamically instantiate the persistence implementation using the service interface through IService. The Factory consists of a default constructor so it can be instantiated by other classes, a shared `get_instance` function based on the

```
Imports Microsoft.VisualBasic
Imports System.Runtime.Serialization

Namespace Domain

<Serializable()> _
Public Class Code
    '*****
    ' CLASS: Code
    ' CREATED BY: Stephen C. Rash
    ' CREATED DATE: 07/20/2009
    ' UPDATED BY: Stephen C. Rash
    ' UPDATED DATE: 07/20/2009
    ' PURPOSE: Holds the code object.
    '*****

    'Variable Declarations
    Private int_code_id As Integer
    Private int_code_type_id As Integer
    Private str_code_name As String
    Private int_code_dep_id As Integer
    Private bol_code_active As Boolean
    Private str_code_lmod_user As String
    Private dte_code_lmod_date As Date
    Private str_code_lmod_type As String

    Public Property code_id() As Integer

        Get
            Return int_code_id
        End Get

        Set(ByVal Value As Integer)
            int_code_id = Value
        End Set

    End Property

    Public Property code_lmod_type() As String

        Get
            Return str_code_lmod_type
        End Get

        Set(ByVal Value As String)
            str_code_lmod_type = Value
        End Set

    End Property

    'Default New Constructor
    Public Sub New()
    End Sub

    'New Overloaded Constructor
    Public Sub New(ByVal code_id As Integer,
        ByVal code_type_id As Integer, ByVal code_name As String,
        ByVal code_dep_id As Integer, ByVal code_active As Boolean,
        ByVal code_lmod_user As String, ByVal code_lmod_date As
        Date, ByVal code_lmod_type As String)

        int_code_id = code_id
        int_code_type_id = code_type_id
        str_code_name = code_name
        int_code_dep_id = code_dep_id
        bol_code_active = code_active
        str_code_lmod_user = code_lmod_user
        dte_code_lmod_date = code_lmod_date
        str_code_lmod_type = code_lmod_type

    End Sub

    'Override ToString
    Public Overrides Function ToString() As String

        Return [String].Format("{0}, {1}, {2}, {3}, {4}, {5}, {6},
        {7}", int_code_id, int_code_type_id, str_code_name,
        int_code_dep_id, bol_code_active, str_code_lmod_user,
        dte_code_lmod_date, str_code_lmod_type)

    End Function

    'Validate Object
    Public Function Validate() As Boolean

        If Not IsNumeric(int_code_id) Then
            Return False
        End If

        If str_code_lmod_type Is Nothing Then
            Return False
        End If

        Return True

    End Function

End Class

End Namespace
```

Object Declarations

Object Properties

Default empty constructor

Overloaded New constructor

Convert To String function

Validation function

Figure 21: Domain Class.

Singleton design pattern to ensure that only one Factory is instantiated and a `get_service` function to return the service implementation of a given service interface.

```
Imports System
Imports System.Text
Imports System.Configuration
Imports System.Collections
Imports System.Collections.Specialized
Imports AAT.Utility.Config
Imports AAT.Utility.Messaging

Namespace Service

    Public Class Factory
        '*****
        ' CLASS: Factory
        ' CREATED BY: Stephen C. Rash
        ' CREATED DATE: 07/07/2009
        ' UPDATED BY: Stephen C. Rash
        ' UPDATED DATE: 07/07/2009
        ' PURPOSE: Factory, to create an instance of an object.
        '*****

        'Default Constructor
        Public Sub New()
        End Sub

        'Singleton design pattern to ensure only one (1) instance of Factory
        Private Shared ftry As New Factory()

        Public Shared Function get_instance() As Factory
            Return ftry
        End Function

        'Get Service Name
        Public Function get_service(ByVal service_name As String) As IService
            Dim tpe As Type
            Dim obj As Object = Nothing
            Dim str_svc_type As String

            Try
                'Try to get Service
                str_svc_type = get_app_setting(service_name)
                tpe = Type.GetType(str_svc_type)
                If tpe IsNot Nothing Then
                    obj = DirectCast(Activator.CreateInstance(tpe), IService)
                End If

                Catch ex As Exception
                    'Throw General Exception
                    log_message(ex.Message.ToString, MsgType.Error)
                End Try

                Return DirectCast(obj, IService)
            End Function
        End Class
    End Namespace
```

Default New constructor

Singleton design pattern

get_service function

Figure 22: Factory Class.

The `IService` class is a marker, or serializable, interface and its purpose is to be a common interface to the service interfaces. It is an empty interface which is inherited by the service interfaces. Figure 23 shows an example `IService` class.

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Collections.Generic

Namespace Service

    Public Interface IService
        '*****
        ' CLASS: IService *
        ' CREATED BY: Stephen C. Rash *
        ' CREATED DATE: 07/07/2009 *
        ' UPDATED BY: Stephen C. Rash *
        ' UPDATED DATE: 07/07/2009 *
        ' PURPOSE: Service superclass. *
        '*****

    End Interface

End Namespace
```

Figure 23: IService (Marker Interface) Class.

The service interface classes are used to instantiate the methods used by the service implementation, so the implementation technology is invisible to the Factory and the higher levels in the model. The service interface classes have modify, select and lookup functions that equate to the same functions in the service implementation. An example of a service interface class is shown in Figure 24.

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Collections
Imports AAT.Domain

Namespace Service

    Public Interface ICodeSvc
        Inherits IService
        '*****
        ' CLASS: ICodeSvc *
        ' CREATED BY: Stephen C. Rash *
        ' CREATED DATE: 07/20/2009 *
        ' UPDATED BY: Stephen C. Rash *
        ' UPDATED DATE: 07/20/2009 *
        ' PURPOSE: Interfaces the Code object. *
        '*****

        'Modify Code
        Function modify_code(ByVal code As Code) As Integer — Modify Code method

        'Select Code
        Function select_code(ByVal code_id As Integer) As Code — Select Code method

        'Lookup Code
        Function lookup_code(ByVal code_list As CodeList) As List(Of CodeList) — Lookup Code method

    End Interface

End Namespace
```

Figure 24: Service Interface Class.

The service implementation classes are where the actual persistence technology resides which is used to move data in and out of the data store. In the example application, a Microsoft

```

Imports Microsoft.VisualBasic
Imports System
Imports System.Collections
Imports System.Collections.Generic
Imports System.Data
Imports System.Data.SqlClient
Imports System.Text
Imports AAT.Utility.Config
Imports AAT.Utility.Messaging
Imports AAT.Domain

Namespace Service

    Public Class CodeSvcSQLImpl
        Implements ICodeSvc
        '*****
        ' CLASS: CodeSvcSQLImpl *
        ' CREATED BY: Stephen C. Rash *
        ' CREATED DATE: 07/20/2009 *
        ' UPDATED BY: Stephen C. Rash *
        ' UPDATED DATE: 07/28/2009 *
        ' PURPOSE: SQL Implementation of the Code object. *
        '*****

        'Modify Code
        Private Function modify_code(ByVal code As Code) As Integer
            Implements ICodeSvc.modify_code

            Dim cn As New SqlConnection(get_connection())
            Dim cmd As New SqlCommand
            Dim int_return As Integer = 0

            Try
                cn.Open()
                cmd.Connection = cn ] Open SQL connection
                cmd.CommandText = get_app_setting("CodeModify") — Return SQL stored procedure name
                cmd.CommandType = CommandType.StoredProcedure
                cmd.Parameters.AddWithValue("@code_id", code.code_id)
                cmd.Parameters.AddWithValue("@code_type_id",
                    code.code_type_id)
                cmd.Parameters.AddWithValue("@code_name", code.code_name)
                If code.code_dep_id = 0 Then
                    cmd.Parameters.AddWithValue("@code_dep_id",
                        System.DBNull.Value)
                Else
                    cmd.Parameters.AddWithValue("@code_dep_id",
                        code.code_dep_id)
                End If
                cmd.Parameters.AddWithValue("@code_active", code.code_active)
                cmd.Parameters.AddWithValue("@code_lmod_user",
                    code.code_lmod_user)
                cmd.Parameters.AddWithValue("@code_lmod_date",
                    code.code_lmod_date)
                cmd.Parameters.AddWithValue("@code_lmod_type",
                    code.code_lmod_type)

                int_return = Convert.ToInt32(cmd.ExecuteScalar()) — Execute Code Modify SQL stored
                procedure and return the Code Id
                of the modified Code record

            Catch ex As SqlException
                int_return = ex.Number * -1
                log_message(ex.Message.ToString, MsgType.Error) ] Catch and log error messages

            Catch ex As Exception
                int_return = -1
                log_message(ex.Message.ToString, MsgType.Error) ]

            Finally
                cn.Close()
                cmd.Dispose() ] Clean-up tasks

            End Try

            Return int_return — Return Code Id of record or error code

        End Function
    End Class
End Namespace

```

Figure 25: Service SQL Implementation Class – Modify Function.

SQL server is the data store, the modify, select and lookup functions will equate to their corresponding stored procedures in the database. The modify function will implement the modify function from the service interface. The function first opens a connection to the SQL database, creates the SQL command and gets the specific modify stored procedure name. Then the function builds and populates the stored procedure parameters in the SQL command based on the passed in object. Next, the modify function executes the SQL command and accepts the return integer value. Any exceptions in that process are captured and stored as the return value then logged to the web server. Once all the processing is complete, the function closes the SQL connection, disposes the SQL command and returns the stored value. Figure 25 shows the modify function of an example service implementation.

```

...
'Select Code
Private Function select_code(ByVal code_id As Integer) As Code
    Implements ICodeSvc.select_code

    Dim code As New Code()
    Dim cn As New SqlConnection(get_connection())
    Dim cmd As New SqlCommand
    Dim dr As SqlDataReader = Nothing

    Try
        cn.Open()
        cmd.Connection = cn ] Open SQL connection
        cmd.CommandText = get_app_setting("CodeSelect") ] Return SQL stored procedure name
        cmd.CommandType = CommandType.StoredProcedure
        cmd.Parameters.AddWithValue("@code_id", code_id) ] Build and populate
        ] SQL stored procedure
        ] parameter
        dr = cmd.ExecuteReader()
        While dr.Read
            code.code_id = dr("code_id")
            code.code_type_id = dr("code_type_id")
            code.code_name = dr("code_name")
            code.code_dep_id = dr("code_dep_id")
            code.code_active = dr("code_active")
            code.code_lmod_user = dr("code_lmod_user")
            code.code_lmod_date = dr("code_lmod_date")
            code.code_lmod_type = dr("code_lmod_type")
        End While
        ] Execute Code Select SQL stored
        ] procedure and put the returned
        ] Code record into a Code object

        Catch ex As SqlException
            log_message(ex.Message.ToString, MsgType.Error)
        Catch ex As Exception
            log_message(ex.Message.ToString, MsgType.Error)
        ] Catch and log error messages

    Finally
        cn.Close()
        cmd.Dispose()
        dr.Close()
    ] Clean-up tasks

    End Try

    Return code ] Return the Code record
End Function
...

```

Figure 26: Service SQL Implementation Class – Select Function.

```

...
'Lookup Code
Private Function lookup_code(ByVal code_list As CodeList) As
    List(Of CodeList) Implements ICodeSvc.lookup_code

    Dim lst_code As New List(Of CodeList)
    Dim cn As New SqlConnection(get_connection())
    Dim cmd As New SqlCommand
    Dim dr As SqlDataReader = Nothing

    Try
        cn.Open()
        cmd.Connection = cn ]- Open SQL connection
        cmd.CommandText = get_app_setting("CodeLookup") ]- Return SQL stored procedure name
        cmd.CommandType = CommandType.StoredProcedure
        cmd.Parameters.AddWithValue("@code_type_id",
            code_list.code_type_name)
        cmd.Parameters.AddWithValue("@code_name",
            code_list.code_name)
        cmd.Parameters.AddWithValue("@code_type_dep_id",
            code_list.code_type_dep_name)
        cmd.Parameters.AddWithValue("@code_dep_id",
            code_list.code_dep_name)
        cmd.Parameters.AddWithValue("@code_active",
            code_list.code_active)
        ]- Build and populate
        ]- SQL stored procedure
        ]- parameters

        dr = cmd.ExecuteReader()
        While dr.Read
            lst_code.Add(New CodeList( _
                dr("code_id"), _
                dr("code_type_name"), _
                dr("code_name"), _
                dr("code_type_dep_name"), _
                dr("code_dep_name"), _
                dr("code_active_name")))
        End While
        ]- Execute Code Lookup SQL stored
        ]- procedure and put the returned
        ]- Code records into a list of CodeList
        ]- objects

        Catch ex As SqlException
            log_message(ex.Message.ToString, MsgType.Error)
        Catch ex As Exception
            log_message(ex.Message.ToString, MsgType.Error)
        ]- Catch and log error messages

        Finally
            cn.Close()
            cmd.Dispose()
            dr.Close()
        ]- Clean-up tasks

    End Try

    Return lst_code ]- Return the Code list

End Function

End Class

End Namespace

```

Figure 27: Service SQL Implementation Class – Lookup Function.

The select function implements the select function from the service interface. The select function opens a connection to the SQL database, creates the SQL command, gets the specific select stored procedure name and builds the SQL data reader to accept the return from the database. Next, the function builds and populates the stored procedure parameter, the specific passed in record key, in the SQL command. Then, the function executes the SQL command to populate the return record into the SQL data reader, which, in turn, populates the object. Any exceptions in that process are captured and logged to the web server. Finally, the function closes

the SQL connection, disposes the SQL command, closes the SQL data reader and returns the object. An example of the select function of a service implementation is shown in Figure 26.

The lookup function implements the lookup function from the service interface. The function first opens a connection to the SQL database, creates the SQL command, gets the specific select stored procedure name and builds the SQL data reader to accept the return from the database. Second, the lookup function builds and populates the stored procedure parameter, the specific passed in object, in the SQL command. Then, the function executes the SQL command to populate the return records into the SQL data reader, which, in turn, populates the list of objects. Any processing exceptions are captured and logged to the web server. Once all the processing is complete, the function closes the SQL connection, disposes the SQL command, closes the SQL data reader and returns the list of objects. Figure 27 shows the lookup function of an example service implementation.

Business Classes.

The Business classes handle the use case workflow. The architecture includes a Manager class, an IManager class, a manager interface for each data function and, at least, one manager implantation for each data function.

The Manager class shown in Figure 28 is used as a communication point to the controllers: it dynamically instantiates the rules implementation using the manager interface through IManager and communicates with the Factory to instantiate the proper service implementation. The Manager consists of a default constructor so it can be instantiated by other classes, a shared get_service function to instantiate the Factory, then use its get_service function to return the implementation of a given interface and a get_manager function to return the manager implementation of a given manager interface.


```

Imports Microsoft.VisualBasic
Imports System
Imports System.Collections.Generic
Imports AAT.Utility.Config
Imports AAT.Utility.Messaging
Imports AAT.Service
Imports AAT.Business

Namespace Business

    Public Class Manager
        '*****
        ' CLASS: Manager
        ' CREATED BY: Stephen C. Rash
        ' CREATED DATE: 07/07/2009
        ' UPDATED BY: Stephen C. Rash
        ' UPDATED DATE: 07/07/2009
        ' PURPOSE: Manager, to create an instance of an object.
        '*****

        'Default Constructor
        Private Sub New()
        End Sub

        'Get Service
        Public Shared Function get_service(ByVal service_name As String)
            As IService

            Dim ftry As New Factory()
            Return ftry.get_service(service_name)

        End Function

        'Get Manager Name
        Public Shared Function get_manager(ByVal manager_name As String)
            As IManager

            Dim tpe As Type
            Dim obj_mgr As Object = Nothing
            Dim str_mgr_type As String = Nothing

            Try
                str_mgr_type = get_app_setting(manager_name)
                tpe = Type.GetType(str_mgr_type)
                If tpe IsNot Nothing Then
                    obj_mgr = DirectCast(Activator.CreateInstance(tpe), IManager)
                End If

            Catch ex As Exception
                'Throw General Exception
                log_message(ex.Message.ToString, MsgType.Error)

            End Try

            Return DirectCast(obj_mgr, IManager)

        End Function

    End Class

End Namespace

```

Default New constructor

get_service function

get_manager function

Figure 28: Manager Class.

```

Imports Microsoft.VisualBasic
Imports System
Imports System.Collections.Generic

Namespace Business

    Public Interface IManager
        '*****
        ' CLASS: IManager
        ' CREATED BY: Stephen C. Rash
        ' CREATED DATE: 07/07/2009
        ' UPDATED BY: Stephen C. Rash
        ' UPDATED DATE: 07/07/2009
        ' PURPOSE: Manager superclass.
        '*****

    End Interface

End Namespace

```

Figure 29: IManager (Marker Interface) Class.

The IManager class is a marker, or serializable, interface and its purpose is to be a common interface to the manager interfaces. It is an empty interface which is inherited by the manager interfaces. Figure 29 shows an example IManager class.

The manager interface classes are used to instantiate the methods used by the implementation. The manager interface classes have modify, select and lookup functions that equate to the same functions in the manager implementation. An example of a manager interfaces class is shown in Figure 30.

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Collections
Imports AAT.Domain

Namespace Business

    Public Interface ICodeMgr
        Inherits IManager
        '*****
        ' CLASS: ICodeMgr *
        ' CREATED BY: Stephen C. Rash *
        ' CREATED DATE: 07/20/2009 *
        ' UPDATED BY: Stephen C. Rash *
        ' UPDATED DATE: 07/20/2009 *
        ' PURPOSE: Interfaces the Code object. *
        '*****

        'Modify Code
        Function modify_code(ByVal code As Code) As Integer — Modify Code method

        'Select Code
        Function select_code(ByVal code_id As Integer) As Code — Select Code method

        'Lookup Code
        Function lookup_code(ByVal code_list As CodeList) As List(Of CodeList) — Lookup Code method

    End Interface

End Namespace
```

Figure 30: Manager Interface Class.

The manager implementation classes are where the use case workflow resides, which is used to control how the methods are executed. The manager implementation is architected with a call to the manager to get the particular service implementation, a function which implements the corresponding function from the manager interface and executes the corresponding service implementation. Figure 31 is an example of a manager implementation class.

```

Imports Microsoft.VisualBasic
Imports System
Imports System.Collections
Imports System.Collections.Generic
Imports System.Text
Imports AAT.Utility.Messaging
Imports AAT.Domain
Imports AAT.Service

Namespace Business

Public Class CodeMgrRuleImpl
    Implements ICodeMgr
    '*****
    ' CLASS: CodeMgrRuleImpl *
    ' CREATED BY: Stephen C. Rash *
    ' CREATED DATE: 07/20/2009 *
    ' UPDATED BY: Stephen C. Rash *
    ' UPDATED DATE: 07/20/2009 *
    ' PURPOSE: Rules Implementation of the Code object. *
    '*****

    'Get Service
    Dim code_svc As ICodeSvc =
        DirectCast(Manager.get_service(GetType(ICodeSvc).Name), ICodeSvc) ]- Get Service

    'Modify Code
    Private Function modify_code(ByVal code As Code) As Integer
        Implements ICodeMgr.modify_code
        Return code_svc.modify_code(code)
    End Function ]- Modify Code method

    'Select Code
    Private Function select_code(ByVal code_id As Integer) As Code
        Implements ICodeMgr.select_code
        Return code_svc.select_code(code_id)
    End Function ]- Select Code method

    'Lookup Code
    Private Function lookup_code(ByVal code_list As CodeList) As
        List(Of CodeList) Implements ICodeMgr.lookup_code
        Return code_svc.lookup_code(code_list)
    End Function ]- Lookup Code method

End Class

End Namespace

```

Figure 31: Manager Rules Implementation Class.

The Controller

The controller is the point of communication between the user interface, the view and the processing, the model in the application. All user commanded actions flow through the controller and the results of those actions are returned to the user by the controller. The controller handles the instantiation of communication channels, user security, object gets and sets, population of lists and grids as well as the actual processing of the user requests.

Controller Classes (Code Behind).

The controller communicates with the business layer by way of the managers. The first step must be to instantiate the communication paths to the managers. Next, the load of the page

calls the security facility, builds any drop-down lists and sets the initial state of the user interface.

Figure 32 shows the page load portion of an example controller class.

```
Imports Microsoft.VisualBasic
Imports Infragistics.Web.UI.LayoutControls
Imports AAT.Domain
Imports AAT.Business
Imports AAT.Utility.Messaging
Imports AAT.Utility.Security
Imports AAT.Utility.General
Imports AAT.Utility.Config

Partial Public Class CodeData
    Inherits System.Web.UI.UserControl
    '*****
    ' CLASS: CodeData *
    ' CREATED BY: Stephen C. Rash *
    ' CREATED DATE: 08/05/2009 *
    ' UPDATED BY: Stephen C. Rash *
    ' UPDATED DATE: 10/27/2009 *
    ' PURPOSE: The Code Data Controller. *
    '*****

    'Get Managers
    Dim app_sec_mgr As IAppSecMgr =
        DirectCast(Manager.get_manager(GetType(IAppSecMgr).Name), IAppSecMgr)
    Dim code_mgr As ICodeMgr =
        DirectCast(Manager.get_manager(GetType(ICodeMgr).Name), ICodeMgr)
    Dim code_type_mgr As ICodeTypeMgr =
        DirectCast(Manager.get_manager(GetType(ICodeTypeMgr).Name),
        ICodeTypeMgr)
    '*****

    'Page Load
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As
        System.EventArgs) Handles Me.Load

        If Not Page.IsPostBack Then

            'Security
            app_security()

            'Search
            populate_ddl_code_type_id()
            populate_ddl_search_active()
            new_code_search()

            'Data
            new_code_data()

        End If

    End Sub
    ...

```

Instantiate Managers

On initial load, run security, any initial drop-down list populations and set the new state for both the search and the data

Figure 32: Controller Class – Page Load.

The security function makes a call to return the level of access the current user has to the particular control. Once the values are returned, they are held in security fields on the control itself for use in setting the view state, allowing the user access to the fields and actions of that control. If the current user has no access to the control, the security fields are set to allow no access to any field or action on the control. The security portion of an example controller class is shown in Figure 33.

```

...
#Region "Security"

Protected Sub app_security()

    Dim lst_app_sec As New List(Of AppSecList)
    Dim app_sec_list As New AppSecList

    'Get Search Criteria
    app_sec_list.app_user = get_current_user()
    app_sec_list.app_page = "%%"
    app_sec_list.app_path = AppRelativeVirtualPath.ToString
    app_sec_list.app_insert = False
    app_sec_list.app_update = False
    app_sec_list.app_delete = False

    'Get List
    lst_app_sec = app_sec_mgr.lookup_app_sec(app_sec_list)

    If lst_app_sec.Count = 0 Then
        hdn_sec_insert.Text = False
        hdn_sec_update.Text = False
        hdn_sec_delete.Text = False
    Else
        app_sec_list = lst_app_sec.Item(0)
        hdn_sec_insert.Text = app_sec_list.app_insert
        hdn_sec_update.Text = app_sec_list.app_update
        hdn_sec_delete.Text = app_sec_list.app_delete
    End If

End Sub

#End Region
...

```

Set security search criteria, current logged on user and current control path

Return security from the database into a list of security

If the list of security has records, update the security fields on the control with the corresponding security, if not default no security

Figure 33: Controller Class – Security.

```

...
#Region "Functions"

'New Search Code
Private Sub new_code_search()

    'Fields
    ddl_search_code_type_id.SelectedValue = "%%"
    txt_search_code_name.Text = ""
    ddl_search_code_active.SelectedValue = "%%"

    'Message
    lbl_gv_message.Text = ""
    img_gv_message.Visible = False
    lbl_gv_message.Visible = False

    'GridView
    gv_code.Visible = False

End Sub

'New Data Code
Private Sub new_code_data()

    'New Record
    ddl_code_type_id.SelectedIndex = 0
    txt_code_name.Text = ""
    populate_ddl_code_dep_id(ddl_code_type_id.SelectedValue)
    If ddl_code_dep_id.Items.Count >= 1 Then
        ddl_code_dep_id.SelectedIndex = 0
    End If
    cbx_code_active.Checked = True

    'Hidden
    hdn_code_id.Text = 0
    hdn_code_lmod_user.Text = "New"
    hdn_code_lmod_date.Text = Now()
    hdn_code_lmod_type.Text = "N"

    'Message
    lbl_data_message.Text = ""
    img_data_message.Visible = False
    lbl_data_message.Visible = False

    'Set View State
    set_view_state()

    'Unselect Index
    gv_code.SelectedIndex = -1

End Sub
...

```

Set initial state of search fields

Set the initial state of the search grid message

Hide the empty search grid

Set initial state of the data fields

Set the initial state of the record key and log fields

Set the initial state of the data message

Set the control view state based on user security

Remove any item selection in the search grid

Figure 34: Controller Class – New Functions.

The initial or new state of the view must be set, with any defaulted values and any displayed or hidden controls. This initial state is also used by the application when the user wants to clear the contents and return to the initial state. Figure 34 shows the new function portion of an example controller class.

Get functions shown in Figure 35 populate an object with the user entered values and pass the object back to the requesting function. The get functions also handle any data translation or clean up while building the objects.

Figure 36 shows the set functions populate the field values with those from an object provided. The set functions also handle any data translation or clean up populating the fields from the object.

The view state uses security to enable fields and display buttons the user has access to or disable fields and hide buttons the user does not. Figure 37 shows the view state functions.

Actions are set in processing functions shown in Figure 38, so they can be called from within the control. There is a corresponding processing function to each button on the view.

To limit values a user can select in the system, drop-down lists are populated from the database. The drop-down lists are populated from a list returned based on criteria provided. The list population functions are shown in Figure 39.

The search facility is a simple grid populated based on the search criteria provided by the user. There are limitations in the number of records that can be successfully returned, the grids are limited to 200 records for processing. If the returned record set is outside the acceptable limits, a message is displayed for the user. The search grid is paging enabled, so the index must be captured as the user moves through the pages. Selected values are populated in the data area

and, based on the record key, provided to the refresh processing function. The search grid population functions are shown in Figure 40.

```

...
'Get Code List
Private Function get_code_list() As CodeList

    Dim code_list As New CodeList — Instantiate CodeList object

    code_list.code_id = 0
    code_list.code_type_name = ddl_search_code_type_id.SelectedValue
    If txt_search_code_name.Text <> "" Then
        code_list.code_name = "%" & txt_search_code_name.Text & "%"
    Else
        code_list.code_name = "%" 'ALL
    End If
    code_list.code_type_dep_name = "%"
    code_list.code_dep_name = "%"
    code_list.code_active = ddl_search_code_active.SelectedValue

    Return code_list — Return the CodeList object

End Function

'Get Code Data
Private Function get_code_data() As Code

    Dim code As New Code — Instantiate Code object

    code.code_id = hdn_code_id.Text
    code.code_type_id = ddl_code_type_id.SelectedValue
    code.code_name = txt_code_name.Text
    If ddl_code_dep_id.SelectedValue <> "" Then
        code.code_dep_id = ddl_code_dep_id.SelectedValue
    Else
        code.code_dep_id = 0
    End If
    code.code_active = cbx_code_active.Checked
    code.code_lmod_user = hdn_code_lmod_user.Text
    code.code_lmod_date = hdn_code_lmod_date.Text
    code.code_lmod_type = hdn_code_lmod_type.Text

    Return code — Return the Code object

End Function
...

```

Populate the CodeList object with the user entered search criteria

Populate the Code object with the user entered Code record data

Figure 35: Controller Class – Get Functions.

```

...
'Set Code Data
Private Sub set_code_data(ByVal code As Code)

    hdn_code_id.Text = code.code_id
    ddl_code_type_id.SelectedValue = code.code_type_id
    txt_code_name.Text = code.code_name
    If code.code_type_id <> 0 Then
        ddl_code_dep_id.SelectedValue = code.code_dep_id
    End If
    cbx_code_active.Checked = code.code_active
    hdn_code_lmod_user.Text = code.code_lmod_user
    hdn_code_lmod_date.Text = code.code_lmod_date
    hdn_code_lmod_type.Text = code.code_lmod_type

End Sub
...

```

Populate the Code data fields with the Code information returned from the database

Figure 36: Controller Class – Set Functions.

```

...
'Set View State
Private Sub set_view_state()

    'Save Button
    If (hdn_sec_insert.Text = True _
        And hdn_code_id.Text = 0) _
        Or (hdn_sec_update.Text = True _
            And hdn_code_id.Text <> 0) Then
        btn_save.Visible = True
    Else
        btn_save.Visible = False
    End If

    'New Button
    If hdn_sec_insert.Text = True Then
        img_new.Visible = True
        btn_new.Visible = True
    Else
        img_new.Visible = False
        btn_new.Visible = False
    End If

    'Refresh Button
    If (hdn_sec_update.Text = True _
        Or hdn_sec_insert.Text = True) _
        And hdn_code_id.Text <> 0 Then
        img_refresh.Visible = True
        btn_refresh.Visible = True
    Else
        img_refresh.Visible = False
        btn_refresh.Visible = False
    End If

    'Delete Button
    If hdn_sec_delete.Text = True _
        And hdn_code_id.Text <> 0 Then
        img_delete.Visible = True
        btn_delete.Visible = True
    Else
        img_delete.Visible = False
        btn_delete.Visible = False
    End If

    'Fields
    If (hdn_sec_insert.Text = True _
        And hdn_code_id.Text = 0) _
        Or (hdn_sec_update.Text = True _
            And hdn_code_id.Text <> 0) Then
        ddl_code_type_id.Enabled = True
        txt_code_name.Enabled = True
        ddl_code_dep_id.Enabled = True
        cbx_code_active.Enabled = True
    Else
        ddl_code_type_id.Enabled = False
        txt_code_name.Enabled = False
        ddl_code_dep_id.Enabled = False
        cbx_code_active.Enabled = False
    End If

End Sub
...

```

Set the state of the buttons based on the user's security

Set the state of the Code data fields based on the user's security

Figure 37: Controller Class – Set View State.

The user is displayed messages about the processing of the data. Both an icon and literal message text are displayed based on the type of message. Figure 41 shows the user messaging functions. Buttons are the actual actions the user can perform from the view. For the most part, the button calls the corresponding processing function, but some have a simple validation to ensure a record key exists before processing. The button functions are shown in Figure 42.


```

...
'Search Code
Private Sub search_code()

    populate_gv_code() — Populate the search grid view

End Sub

'Refresh Code
Private Sub refresh_code()

    Dim code As New Code — Instantiate Code object

    'Select code
    code = code_mgr.select_code(hdn_code_id.Text) — Populate the Code object with the return from
                                                the database based on the Code record key

    'Get Code Dependency
    If code.code_type_id <> 0 Then
        populate_ddl_code_dep_id(code.code_type_id)
    Else
        ddl_code_dep_id.Enabled = False
    End If
    ] — Reset drop-down list or disable

    'Set Code
    set_code_data(code) — Set the Code data fields with the returned Code object

    'Message
    lbl_data_message.Text = ""
    img_data_message.Visible = False
    lbl_data_message.Visible = False ] — Reset the state of the data message

    'Set View State
    set_view_state() ————— Set the control view state based on user security

End Sub

'Modify Code
Private Sub modify_code(ByVal type As ModType)

    Dim int_ret As Integer = 0
    Dim code As New Code — Instantiate Code object
    Dim str_msg As String

    'Get Code
    code = get_code_data() — Populate the Code object with the Code data fields

    'Set Last Mod
    code.code_lmod_user = get_current_user()
    code.code_lmod_date = Now()
    If type = ModType.Delete Then
        code.code_lmod_type = "D"
        str_msg = "Delete"
    Else
        If code.code_id = 0 Then
            code.code_lmod_type = "I"
            str_msg = "Insert"
        Else
            code.code_lmod_type = "U"
            str_msg = "Update"
        End If
    End If
    ] — Set the log fields based on the current
        user, current date/time and the origin of
        the modification call

    End If

    'Modify
    int_ret = code_mgr.modify_code(code) — Send the Code record to the database and get returned
                                        Code record key or error code

    'Complete Message
    If int_ret > 0 Then
        If type <> ModType.Delete Then
            hdn_code_id.Text = int_ret
            refresh_code()
        Else
            new_code_data()
            populate_gv_code()
        End If
        str_msg = str_msg & " Successful."
        data_message(MsgType.Info, str_msg)
    Else
        str_msg = str_msg & " FAILED: " & get_app_setting(int_ret)
        data_message(MsgType.Error, str_msg)
    End If
    ] — Display a message to the
        user if the operation was
        successful, a Code record
        key was returned or failed,
        an error code was returned

End Sub
...

```

Figure 38: Controller Class – Processing Functions.

```

...
'Populate Code Type DDL
Private Sub populate_ddl_code_type_id()
    ...
End Sub

'Populate Code Dependency DDL
Private Sub populate_ddl_code_dep_id(ByVal str_code_type As String)

    Dim lst_code As New List(Of CodeList)
    Dim code_list As New CodeList
    Dim code_type As New CodeType
    ] Instantiate the CodeList, CodeType and
    list of CodeList objects

    'Get Code Type Data
    code_type = code_mgr.select_code_type(str_code_type)
    — Populate the CodeType object
    with the return from the database
    based on the code type field

    'Build Selection
    code_list.code_id = 0
    code_list.code_type_name = code_type.code_type_dep_id
    code_list.code_name = "%s"
    code_list.code_type_dep_name = "%s"
    code_list.code_dep_name = "%s"
    code_list.code_active = "%s"
    ] Populate the CodeList object with
    the criteria for the drop-down list

    'Get List
    lst_code = code_mgr.lookup_code(code_list)
    — Populate the list of CodeList object with the
    return from the database based on the
    CodeList object

    'Populate List
    ddl_code_dep_id.DataSource = lst_code
    ddl_code_dep_id.DataTextField = "code_name"
    ddl_code_dep_id.DataValueField = "code_id"
    ddl_code_dep_id.DataBind()
    ] Populate the drop-down list with the
    returned list of CodeList object

    If lst_code.Count > 0 Then
        ddl_code_dep_id.Enabled = True
    Else
        ddl_code_dep_id.Enabled = False
    End If
    ] Enable or disable based on count of the
    list of Code List object

End Sub

'Refresh on Change
Protected Sub ddl_code_type_id_SelectedIndexChanged(ByVal sender As
Object, ByVal e As EventArgs) Handles
ddl_code_type_id.SelectedIndexChanged

    populate_ddl_code_dep_id(ddl_code_type_id.SelectedValue)
    — Repopulate the drop-down list
    on change in code type

End Sub

'Populate Code Active DDL
Private Sub populate_ddl_search_active()

    ddl_search_code_active.Items.Add(New ListItem("Active Only", "1"))
    ddl_search_code_active.Items.Add(New ListItem("Inactive Only", "0"))
    ddl_search_code_active.Items.Add(New ListItem("<ALL>", "%s"))
    ] Manually add values to
    a drop-down list

End Sub
...

```

Figure 39: Controller Class – Drop-down Lists.

The controller is architected in a simple way to facilitate the seamless communication between the view and the model.

Utility Classes.

The utility classes are used to hold global functions that allow for reuse. The functions within the utility classes are shared, so they can be used by any of the controllers to perform common or repetitive functions. Figure 43 shows an example utility class.

```

...
'Populate Code Grid View
Private Sub populate_gv_code()

    Dim lst_code As New List(Of CodeList) ] Instantiate the CodeList and list of CodeList objects
    Dim code_list As New CodeList

    'Get Search Criteria
    code_list = get_code_list() — Populate the CodeList object with the Code search criteria

    'Get List
    lst_code = code_mgr.lookup_code(code_list) — Populate the list of CodeList object with
    the return from the database based on the
    CodeList object

    'Set Grid Values / Messages
    If lst_code.Count = 0 Then
        img_gv_message.Visible = True
        lbl_gv_message.Visible = True
        lbl_gv_message.Text = "No Records Returned!"
        gv_code.Visible = False
    ElseIf lst_code.Count = 201 Then
        img_gv_message.Visible = True
        lbl_gv_message.Visible = True
        lbl_gv_message.Text = "Too Many Records to Return,
        Please Limit Search!"
        gv_code.Visible = False
    Else
        img_gv_message.Visible = False
        lbl_gv_message.Visible = False
        gv_code.DataSource = lst_code
        gv_code.Columns(1).Visible = True
        gv_code.DataBind()
        gv_code.Columns(1).Visible = False 'Hide Key Field
    End If

    'Unselect Index
    gv_code.SelectedIndex = -1 — Remove any item selection in the search grid

End Sub

'Code Grid View Paging
Private Sub gv_code_PageIndexChanging(ByVal sender As Object, ByVal e As
System.Web.UI.WebControls.GridViewPageEventArgs) Handles
gv_code.PageIndexChanging

    'Change Page Index
    gv_code.PageIndex = e.NewPageIndex — Set the new page index

    'Repopulate Grid View
    populate_gv_code() — Repopulate the search grid view

End Sub

'Code Grid View Selection
Protected Sub gv_code_SelectedIndexChanged(ByVal sender As Object,
ByVal e As EventArgs) Handles gv_code.SelectedIndexChanged

    'Get Selected Key
    hdn_code_id.Text = gv_code.SelectedRow.Cells(1).Text — Set the Code record key with the
    selected Code record key

    'Refresh Data
    refresh_code() — Refresh the Code record

End Sub
...

```

Figure 40: Controller Class – Search Grid.

The View

The view is the user interface, the representation of the application presented to the users. The view is where the user performs tasks necessary to their specific job function. The view consists of the master pages, pages, user controls, as well as, images and cascading style sheets that control the look and feel to the user.

```

...
'Data Messages
Private Sub data_message(ByVal msg_type As MsgType, ByVal msg_text As
String)

    'Set Message
    lbl_data_message.Text = msg_text — Set the data message text with the passed string

    'Set Images / Text
    If msg_type = MsgType.Error Then
        img_data_message.ImageUrl = "~/Images/Messages/msg_warning.gif"
        lbl_data_message.CssClass = "msg-negative"
    ElseIf msg_type = MsgType.Warning Then
        img_data_message.ImageUrl = "~/Images/Messages/msg_caution.gif"
        lbl_data_message.CssClass = "msg-negative"
    ElseIf msg_type = MsgType.Info Then
        img_data_message.ImageUrl = "~/Images/Messages/msg_information.gif"
        lbl_data_message.CssClass = "msg-positive"
    ElseIf msg_type = MsgType.Question Then
        img_data_message.ImageUrl = "~/Images/Messages/msg_question.gif"
        lbl_data_message.CssClass = "msg-neutral"
    End If

    'Display Message
    img_data_message.Visible = True — Display the data message image and data message text
    lbl_data_message.Visible = True

End Sub

#End Region
...

```

Figure 41: Controller Class – Messages.

```

...
#Region "Buttons"

'Search Button
Protected Sub btn_search_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs) Handles btn_search.Click

    'Clear Grid Message
    lbl_gv_message.Text = ""
    img_gv_message.Visible = False — Reset the state of the search grid message
    lbl_gv_message.Visible = False

    search_code() — Search for Codes

End Sub

'Reset Search
Protected Sub btn_reset_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs) Handles btn_reset.Click

    new_code_search() — Reset Code search criteria

End Sub

'Save Button
Protected Sub btn_save_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs) Handles btn_save.Click

    modify_code(ModType.Save) — Modify the Code record with a type of Save

End Sub

'New Button
Protected Sub btn_new_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs) Handles btn_new.Click

    new_code_data() — Reset the Code data fields

End Sub

'Refresh Button
Protected Sub btn_refresh_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs) Handles btn_refresh.Click

    'Exit if not a valid key
    If hdn_code_id.Text = 0 Then
        Exit Sub
    Else
        refresh_code() — If the Code record key is not 0, new record then
        refresh the Code record
    End If

End Sub

>Delete Button
Protected Sub btn_delete_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs) Handles btn_delete.Click

    'Exit if not a valid key
    If hdn_code_id.Text = 0 Then
        Exit Sub
    Else
        modify_code(ModType.Delete) — If the Code record key is not 0, new record then
        modify the Code record with a type of Delete
    End If

End Sub

#End Region

```

Figure 42: Controller Class – Buttons.

```

Imports Microsoft.VisualBasic

Namespace Utility

Public Class Config
'*****
' CLASS: Config
' CREATED BY: Stephen C. Rash
' CREATED DATE: 07/07/2009
' UPDATED BY: Stephen C. Rash
' UPDATED DATE: 07/09/2009
' PURPOSE: Handles web.config access activities.
'*****

'Get Environment
Public Shared Function get_environment() As String

    Dim str_env As String = get_app_setting("Environment")
    Return str_env
} Returns the specific Environment
setting value

End Function

'Get Version
Public Shared Function get_version() As String

    Dim str_env As String = get_app_setting("Version")
    Return str_env
} Returns the specific Version
setting value

End Function

'Get Connection String
Public Shared Function get_connection() As String

    Dim str_env As String = get_environment()
    Dim str_conn As String =
        ConfigurationManager.ConnectionStrings(str_env).ToString
    Return str_conn
} Returns the connection string
based on the Environment
setting value

End Function

'Get Implementation Name
Public Shared Function get_app_setting(ByVal str_name As String) As
String

    Dim settings As NameValueCollection =
        ConfigurationManager.AppSettings
    Return settings.Get(str_name)
} Returns a general setting value based on
a passed in key name

End Function

End Class

End Namespace

```

Figure 43: Utility Class.

User Controls.

The user control is the main piece of the view. The user functions are encapsulated in user controls to allow for reuse, and contain all the functionality of the application. The example application is data driven, so each function must allow for the basic data functions to search, insert, update and delete data records.

The first part of the user control is a search facility to accept user input to limit the returned subset of records. Example search criteria fields' code is shown in Figure 44.

```

</%> Control
Language="vb"
AutoEventWireup="false"
CodeBehind="CodeData.ascx.vb"
Inherits="AAT.CodeData"
%>

<table>
  <tr>
    <td align="left" colspan="2">
      <h3>Code Maintenance</h3> — Displayed header
    </td>
  </tr>
  ...
  <tr>
    <td align="right">
      <asp:Label
        id="lbl_search_code_name"
        runat="server"
        CssClass="field-nonrequired">Code</asp:Label>
      ] Search label with style class
      defined
    </td>
    <td align="left">
      <asp:TextBox
        id="txt_search_code_name"
        runat="server"
        MaxLength="50"
        Width="200px"></asp:TextBox>
      ] Search field
    </td>
  </tr>
  ...

```

Figure 44: View – Search Criteria Fields.

Next, Figure 45 shows the search action buttons that affect the search and reset the search criteria fields if necessary. Any search related messages are displayed to the user.

```

...
<tr>
  <td align="center" colspan="2">
    <asp:ImageButton
      id="btn_search"
      runat="server"
      ImageUrl="~/Images/Buttons/btn_search.gif"
      CausesValidation="False" />
    <asp:Image
      id="img_reset"
      runat="server"
      ImageUrl="~/Images/Buttons/btn_xsp.gif" />
    <asp:ImageButton
      id="btn_reset"
      runat="server"
      CausesValidation="False"
      ImageUrl="~/Images/Buttons/btn_reset.gif" />
    ] Search button
    ] Spacer image
    ] Search reset button
  </td>
</tr>
<tr>
  <td align="center" colspan="2">
    <asp:Image
      ID="img_gv_message"
      runat="server"
      ImageUrl="~/Images/Messages/msg_caution.gif" />
    <asp:Label
      ID="lbl_gv_message"
      runat="server"
      CssClass="msg-negative"></asp:Label>
    ] Search grid message image
    ] Search grid message text
  </td>
</tr>
...

```

Figure 45: View – Search Actions.

The search grid displays the returned records from the database in such a way that the user can select the record needed. By selecting a record, the data is populated in the data area for modification. Figure 46 shows code for the search grid controls.

```

...
<tr>
  <td align="center" colspan="2">
    <asp:GridView
      ID="gv_code"
      runat="server"
      AllowPaging="True"
      AutoGenerateColumns="False">
      <PagerSettings Mode="NextPreviousFirstLast"
        FirstPageImageUrl="~/Images/Other/page_first.gif"
        FirstPageText="First"
        LastPageImageUrl="~/Images/Other/page_last.gif"
        LastPageText="Last"
        NextPageImageUrl="~/Images/Other/page_next.gif"
        NextPageText="Next"
        PreviousPageImageUrl="~/Images/Other/page_previous.gif"
        PreviousPageText="Previous" />
      <RowStyle CssClass="grid-row" />
      <Columns>
        <asp:CommandField
          ButtonType="Image"
          SelectImageUrl="~/Images/Other/list_select.gif"
          ShowSelectButton="True" />
        <asp:BoundField
          DataField="code_id"
          HeaderText="Code" />
        <asp:BoundField
          DataField="code_type_name"
          HeaderText="Code Type" />
        <asp:BoundField
          DataField="code_name"
          HeaderText="Code" />
        <asp:BoundField
          DataField="code_active"
          HeaderText="Active"
          />
      </Columns>
      <FooterStyle CssClass="grid-footer" />
      <PagerStyle CssClass="grid-pager" />
      <SelectedRowStyle CssClass="grid-selected" />
      <HeaderStyle CssClass="grid-header" />
    </asp:GridView>
  </td>
</tr>
<tr>
  <td align="center" colspan="2">
    <hr style="color:Maroon; size:1; height: 1px;" />
  </td>
</tr>
...

```

Figure 46: View – Search Grid.

The data action buttons are next. Figure 47 shows the code to setup of the action buttons. They facilitate the inserting, updating or deleting of the selected data record. Any data related messages are displayed to the user. The validation summary displays a literal message to the user when there are data field validation errors. The validation summary code is shown in Figure 48. The data fields hold the data record information to be manipulated by the user. They are

displayed as a label and a field. Any validations are attached to the field to display when a validation error occurs. The code for the data fields is shown in Figure 49.

```

...
<tr>
  <td align="center" colspan="2">
    <asp:ImageButton
      id="btn_save"
      runat="server"
      CausesValidation="True"
      ImageUrl="~/Images/Buttons/btn_save.gif"
      ValidationGroup="Code" />
    <asp:Image
      id="img_new"
      runat="server"
      ImageUrl="~/Images/Buttons/btn_xsp.gif" />
    <asp:ImageButton
      id="btn_new"
      runat="server"
      CausesValidation="False"
      ImageUrl="~/Images/Buttons/btn_new.gif" />
    <asp:Image
      id="img_refresh"
      runat="server"
      ImageUrl="~/Images/Buttons/btn_xsp.gif" />
    <asp:ImageButton
      id="btn_refresh"
      runat="server"
      CausesValidation="False"
      ImageUrl="~/Images/Buttons/btn_refresh.gif" />
    <asp:Image
      id="img_delete"
      runat="server"
      ImageUrl="~/Images/Buttons/btn_xsp.gif" />
    <asp:ImageButton
      id="btn_delete"
      runat="server"
      CausesValidation="False"
      ImageUrl="~/Images/Buttons/btn_delete.gif" />
  </td>
</tr>
<tr>
  <td align="center" colspan="2">
    <asp:Image
      ID="img_data_message"
      runat="server"
      ImageUrl="~/Images/Buttons/msg_information.gif" />
    <asp:Label
      ID="lbl_data_message"
      runat="server"
      CssClass="msg-negative"></asp:Label>
  </td>
</tr>
...

```

Figure 47: View – Data Actions.

```

...
<tr>
  <td align="left" colspan="2">
    <asp:ValidationSummary
      ID="vsm_data"
      runat="server"
      CssClass="msg-validation"
      HeaderText="The following errors were encountered:"
      ValidationGroup="Code" />
  </td>
</tr>
...

```

Figure 48: View – Validation Summary.


```

...
<tr>
  <td align="right">
  </td>
  <td align="left">
    <asp:TextBox
      id="hdn_code_id"
      runat="server"
      Visible="False"
      BackColor="#FFFF99"></asp:TextBox>
  </td>
</tr>
<tr>
  <td align="right">
    <asp:label
      id="lbl_code_type_id"
      runat="server"
      CssClass="field-required">Code Type</asp:label>
  </td>
  <td align="left">
    <asp:DropDownList
      id="ddl_code_type_id"
      runat="server"
      AutoPostBack="True"></asp:DropDownList>
    <asp:RequiredFieldValidator
      id="rfv_code_type_id"
      runat="server"
      ErrorMessage="Code Type is Required!"
      ControlToValidate="ddl_code_type_id"
      ValidationGroup="Code">*</asp:RequiredFieldValidator>
  </td>
</tr>
...

```

Hidden Code key field

Data field label with style class defined

Data field (drop-down list)

Data field validation control, linked to specific data field with validation group defined

Figure 49: View – Data Fields.

Hidden fields hold any data for processing or security that the application needs to hold, but not display to the user. Figure 50 shows the code for the hidden field setup.

```

...
<tr>
  <td colspan="2" align="center">
    <asp:TextBox
      id="hdn_code_lmod_user"
      runat="server"
      Visible="False"
      BackColor="#FFFF99"></asp:TextBox>
    <asp:TextBox
      id="hdn_code_lmod_date"
      runat="server"
      Visible="False"
      BackColor="#FFFF99"></asp:TextBox>
    <asp:TextBox
      id="hdn_code_lmod_type"
      runat="server"
      Visible="False"
      BackColor="#FFFF99"></asp:TextBox>
  </td>
</tr>
<tr>
  <td colspan="2" align="center">
    <asp:TextBox
      id="hdn_sec_insert"
      runat="server"
      Visible="False"
      BackColor="#CCFFCC"></asp:TextBox>
    <asp:TextBox
      id="hdn_sec_update"
      runat="server"
      Visible="False"
      BackColor="#CCFFCC"></asp:TextBox>
    <asp:TextBox
      id="hdn_sec_delete"
      runat="server"
      Visible="False"
      BackColor="#CCFFCC"></asp:TextBox>
  </td>
</tr>
</table>

```

Hidden log user field

Hidden log date field

Hidden log type field

Hidden insert security field

Hidden update security field

Hidden delete security field

Figure 50: View – Hidden Fields.

Figure 51 shows the view displayed to the user is simple and clean with everything laid out in a logical way.

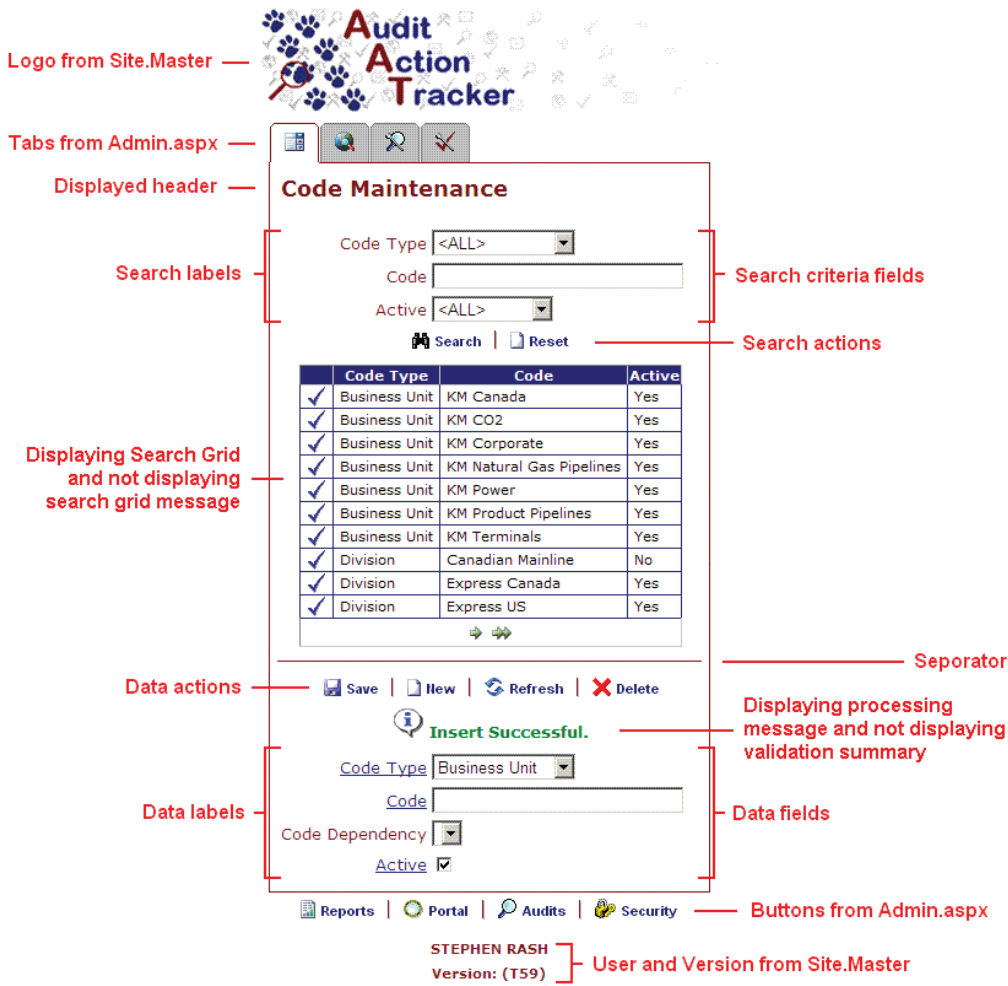


Figure 51: View – User Display.

All together, the view is what the user will see and use to work with the application; if it is not straight forward; easy to use and understand, it will not be a successful application.

Cascading Style Sheets.

Cascading style sheets are a global way to control the look and feel of the application. Building the style of the application in a common place to be pulled anywhere it is needed, is an advantage; if the style changes, that change can be made in one location and affects the entire application. Example code for creating a cascading style sheet is shown in Figure 52.

```

a
{
  text-decoration:none; — Controls the style of any link or 'a' HTML tags
}

body
{
  background-color: white;
  font-family: verdana, helvetica, arial;
  font-weight: normal;
  font-size: small;
  color: navy;
}

table
{
  border-style: none; — Controls the style of any 'table' HTML tags
}

td
{
  font-family: verdana, helvetica, arial;
  font-size: 10pt; — Controls the style of any table detail or 'td' HTML tags
}

h1,h2,h3,h4,h5
{
  font-weight: bold; — Controls the style of any header, 'h1', 'h2', 'h3', 'h4' or 'h5' HTML tags
  color: Maroon;
}
...

```

Figure 52: Cascading Style Sheet.

If the styles of the required and non-required fields change in the application, they can be easily modified in one location. Figure 53 shows code and example the before the change. Change the style sheet class and the change is effective on all the required and non-required fields without having to go to each one and make the change there. Figure 54 shows code and example the after the change.

```

.field-required
{
  /*color: maroon;*/
  text-decoration: underline;
  /*font-style:italic;*/
  text-align: right;
  vertical-align: middle;
}

.field-nonrequired
{
  color: maroon;
  text-align: right;
  vertical-align: middle;
}

```

Required = Navy text (default), underlined

Nonrequired = Maroon text

Figure 53: Changing a Cascading Style Sheet – Before.

```

.field-required
{
  color: maroon;
  text-decoration: underline;
  font-style:italic;
  text-align: right;
  vertical-align: middle;
}

.field-nonrequired
{
  /*color: maroon;*/
  text-align: right;
  vertical-align: middle;
}

```

Required = Maroon text, underline and italic

Nonrequired = Navy text (default)

Figure 54: Changing a Cascading Style Sheet – After.

Validations.

To ensure valid data is entered in the application, field validations are necessary. Visual Studio has delivered validation controls that are easy to use and give the user instant feedback at the point of entry. There are two parts to the validation: a validation summary, which displays the literal validation message and, the specific validation, which assesses a particular data field for validity. The example code for setting up validations is shown in Figure 55.

```

...
<tr>
  <td align="left" colspan="2">
    <asp:ValidationSummary
      ID="vsm_data"
      runat="server"
      CssClass="msg-validation"
      HeaderText="The following errors were encountered:"
      ValidationGroup="Code" />
  </td>
</tr>
...
<tr>
  <td align="right">
    <asp:label
      id="lbl_code_name"
      runat="server"
      CssClass="field-required">Code</asp:label>
  </td>
  <td align="left">
    <asp:TextBox
      id="txt_code_name"
      runat="server"
      MaxLength="50"
      style="width: 200px"></asp:TextBox>
    <asp:RequiredFieldValidator
      id="rfv_code_name"
      runat="server"
      ErrorMessage="Code is Required!"
      ControlToValidate="txt_code_name"
      ValidationGroup="Code">*</asp:RequiredFieldValidator>
  </td>
</tr>
...

```

Validation summary with validation group defined

Data field validation control, linked to specific data field with validation group defined

Figure 55: Validation Code.

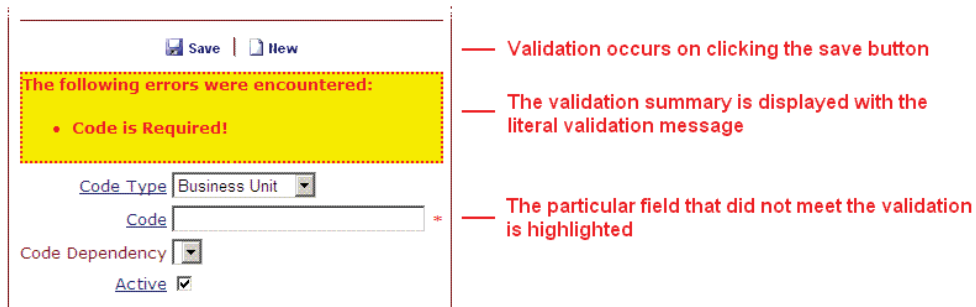


Figure 56: Validation Display.

If there is a validation error, the validation summary is displayed and the field is highlighted that does not meet the validation criteria. Figure 56 shows the validation display.

Messaging.

It is important to inform the user about processing successes and failures. The messaging feature allows the user to see when the process is complete and, if it was successful, or if there was an error. The messaging facility also masks the particular error message returned and displays a more useful message to the user. Figure 57 shows the code for setting up messaging.

```

...
'Modify
int_ret = code_mgr.modify_code(code) — Send the Code record to the database and get returned
Code record key or error code

'Complete Message
If int_ret > 0 Then
    If type <> ModType.Delete Then
        hdn_code_id.Text = int_ret
        refresh_code()
    Else
        new_code_data()
        populate_gv_code()
    End If
    str_msg = str_msg & " Successful."
    data_message(MsgType.Info, str_msg)
Else
    str_msg = str_msg & " FAILED: " & get_app_setting(int_ret)
    data_message(MsgType.Error, str_msg)
End If
End Sub
...
'Data Messages
Private Sub data_message(ByVal msg_type As MsgType, ByVal msg_text As
String)

'Set Message
lbl_data_message.Text = msg_text — Set the data message text with the passed string

'Set Images / Text
If msg_type = MsgType.Error Then
    img_data_message.ImageUrl = "~/Images/Messages/msg_warning.gif"
    lbl_data_message.CssClass = "msg-negative"
ElseIf msg_type = MsgType.Warning Then
    img_data_message.ImageUrl = "~/Images/Messages/msg_caution.gif"
    lbl_data_message.CssClass = "msg-negative"
ElseIf msg_type = MsgType.Info Then
    img_data_message.ImageUrl = "~/Images/Messages/msg_information.gif"
    lbl_data_message.CssClass = "msg-positive"
ElseIf msg_type = MsgType.Question Then
    img_data_message.ImageUrl = "~/Images/Messages/msg_question.gif"
    lbl_data_message.CssClass = "msg-neutral"
End If

'Display Message
img_data_message.Visible = True — Display the data message image and data message text
lbl_data_message.Visible = True

End Sub
#End Region
...

```

Figure 57: Messaging Code.

Based on the message type, the icon and message text are displayed in such a way the user knows if the processing was successful or if it failed. Examples of the display of the messages is shown in Figure 58.

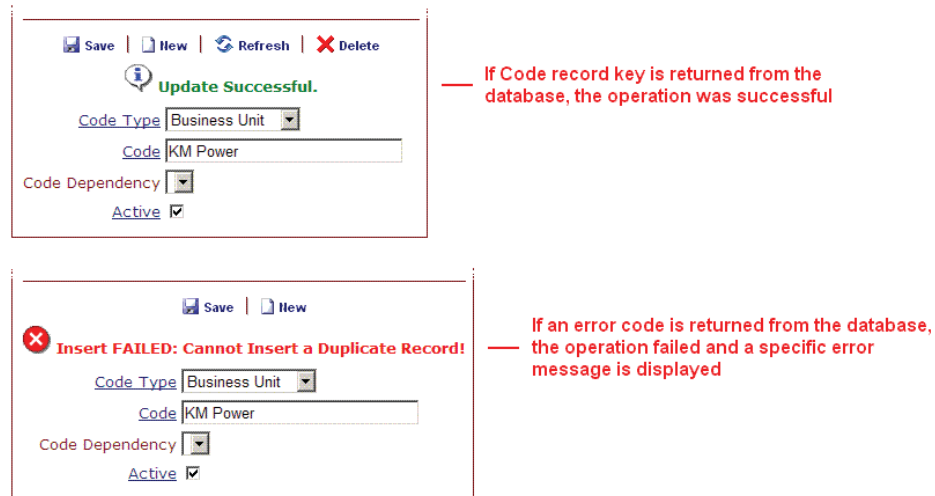


Figure 58: Messaging Display.

Popups.

A modal popup window can be very useful for displaying a message that a user has to acknowledge or to do external processing prior to continuing. The example application uses the modal popup window to lookup values in the system to assign to the parent control. There are many third-party built controls that can be used, but have a lot of processing overhead to deal with. A simple modal popup can be built within the context of the existing tools.

To create the simple modal popup, a cascading style sheet for the popup and modal styles must be created. Figure 59 shows the cascading style sheet code for the simple popup. Next, panels for both the modal and popup must be created. The modal panel has no content; it is just a barrier between the original screen and the popup, so no processing can be accomplished on the original window without first closing the popup. The popup has the content to be processed.

Code to create and display the simple popup is shown in Figure 60.

```

.popup
{
    background-color: White;
    padding-top: 5px;
    padding-right: 5px;
    padding-bottom: 5px;
    padding-left: 5px;
    border-color: Navy;
    border-width: 1px;
    border-style: solid;
    text-align: center;
    position: fixed;
    left: 5%;
    top: 5%;
    z-index: 2;
}

.modal
{
    background-color: White;
    filter: alpha(opacity : 50);
    opacity: 0.5;
    border-color: White;
    border-width: 1px;
    border-style: solid;
    text-align: center;
    width: 100%;
    height: 100%;
    position: fixed;
    left: 1px;
    top: 1px;
    z-index: 1;
}
...

```

Figure 59: Popup Cascading Style Sheet.

```

...
<asp:Panel
    ID="pnl_modal"
    runat="server"
    CssClass="modal"
    HorizontalAlign="Center"
    Visible="false">
</asp:Panel>
<asp:Panel
    ID="pnl_employee_lookup"
    runat="server"
    CssClass="popup"
    HorizontalAlign="Center"
    Visible="false">
    <table>
        <tr>
            <td>
                <asp:UpdatePanel
                    ID="upd_employee_lookup"
                    runat="server"
                    UpdateMode="Conditional">
                    <ContentTemplate>
                        <uc1:Employees ID="Employees1" runat="server" />
                    </ContentTemplate>
                </asp:UpdatePanel>
            </td>
        </tr>
        <tr>
            <td align="center">
                <asp:ImageButton
                    ID="btn_employee_return"
                    runat="server"
                    CausesValidation="False"
                    ImageUrl="~/Images/Buttons/btn_return.gif" />
            </td>
        </tr>
    </table>
</asp:Panel>

```

Figure 60: Popup View Code.

By making both the modal and popup panels visible, the popup is the only item on the screen that can be accessed. By returning the values from the popup and hiding both the modal and popup panels, the original screen is active and has the selected values. Figure 61 shows the code to return values from the popup and hide the panels. The display to the user is simple, the popup, the modal panel between it and the original screen. The display of what the simple popup looks like is shown in Figure 62.

```

...
'Lookup Employee
Protected Sub btn_lookup_usr_user_Click(ByVal sender As Object, ByVal e
  As System.Web.UI.ImageClickEventArgs) Handles btn_lookup_usr_user.Click

    pnl_modal.Visible = True
    pnl_employee_lookup.Visible = True ]- Display the modal panel and popup panel

End Sub

'Return Employee
Protected Sub btn_employee_return_Click(ByVal sender As Object, ByVal e
  As System.Web.UI.ImageClickEventArgs) Handles btn_employee_return.Click

    lkp_usr_user.Text = Employees1.hold_empl_user
    txt_usr_name.Text = Employees1.hold_empl_name ]- Return employee user and name information
    pnl_employee_lookup.Visible = False
    pnl_modal.Visible = False ]- Hide the modal panel and popup panel

End Sub
...

```

Figure 61: Popup Controller Code.

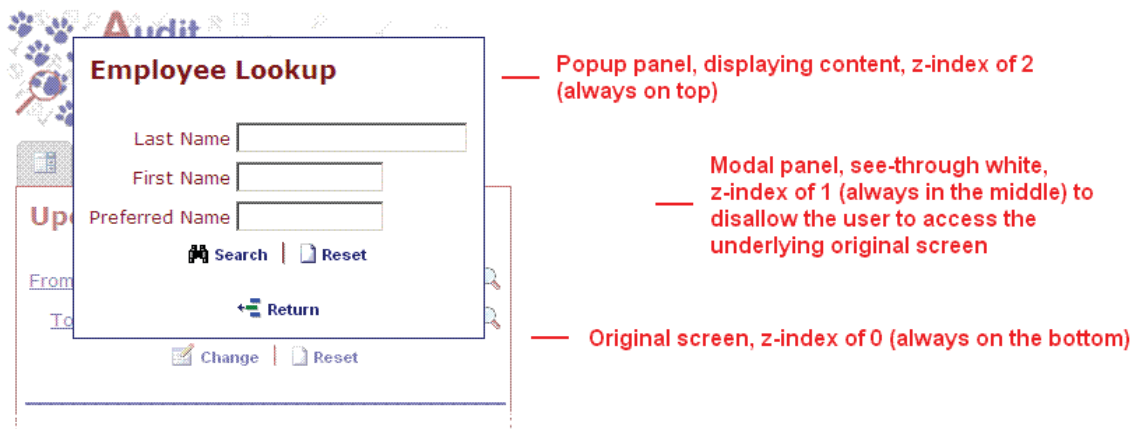


Figure 62: Popup Display.

The Configuration File

The configuration file, the web.config file, is a file that resides outside the application where variables can be assigned and changed without having to republish the application. There are three main areas that are used in the configuration file: the application settings, the connection strings and the authentication section.

The application settings are value pairs stored in the file, which can be changed if necessary to change the returned value to the system. All the processing specific values are stored here, system specific used for display in the application, service and business implementations used in processing and any other variables that could change, such as: database stored procedure names, file locations and even literal error messages. Figure 63 shows the example application setting value pairs held in the configuration file.

```

<?xml version="1.0"?>
<configuration>
...
<appSettings>
  <!--SYSTEM-->
  <add key="Debug" value="False" />
  <!--value="True"/-->
  <add key="Version" value="(T62)" />
  <add key="BuildDate" value="10/21/2009 4:54 PM MST" />
  <add key="Environment" value="Test" />
  <!--EXTERNAL LINKS-->
  <add key="Security" value="~/Pages/Security.aspx" />
  <add key="Admin" value="~/Pages/Admin.aspx" />
  <add key="Audit" value="~/Pages/Audit.aspx" />
  <add key="Assignee" value="~/Pages/Assignee.aspx" />
  <add key="Reports"
    value="http://report/businessobjects/main.aspx" />
  <add key="Portal"
    value="http://portal/ehs/apps/aat/default.aspx" />
  <!--SERVICES-->
  <add key="ICodeTypeSvc" value="AAT.Service.CodeTypeSvcSQLImpl" />
  <add key="ICodeSvc" value="AAT.Service.CodeSvcSQLImpl" />
  ...
  <add key="IUserSvc" value="AAT.Service.UserSvcSQLImpl" />
  <add key="IAppSecSvc" value="AAT.Service.AppSecSvcSQLImpl" />
  <!--RULES-->
  <add key="ICodeTypeMgr" value="AAT.Business.CodeTypeMgrRuleImpl" />
  <add key="ICodeMgr" value="AAT.Business.CodeMgrRuleImpl" />
  ...
  <add key="IUserMgr" value="AAT.Business.UserMgrRuleImpl" />
  <add key="IAppSecMgr" value="AAT.Business.AppSecMgrRuleImpl" />
  <!--PROCEDURES-->
  <add key="CodeTypeSelect" value="sp_code_type_select" />
  <add key="CodeTypeLookup" value="sp_code_type_lookup" />
  ...
  <add key="UserLookup" value="sp_user_lookup" />
  <add key="AppSecLookup" value="sp_app_sec_lookup" />
  <!--UPLOAD FILE LOCATIONS-->
  <add key="TestPath" value="\\test\AAT_Uploads\" />
  <add key="ProdPath" value="\\prod\AAT_Uploads\" />
  <!--ERROR MESSAGES-->
  <add key="-2601" value="Cannot Insert a Duplicate Record!" />
  <add key="-4864" value="Upload File Format Incorrect!" />
</appSettings>
...

```

Figure 63: Configuration File – Application Settings.

Connection strings to the database instances are held in the configuration file and could be changed, if the need arose. Multiple instances can be controlled by an application setting as to which one to use. Figure 64 shows the value pairs of connection strings.

```

...
<connectionStrings>
  <!--TEST-->
  <add name="Test"
    providerName="System.Data.SqlClient"
    connectionString="Data Source=test\test;User
      ID=AATAPP;Pwd=XXXXXXXXXX;Initial Catalog=AAT;" />
  <!--PROD-->
  <add name="Prod"
    providerName="System.Data.SqlClient"
    connectionString="Data Source=prod\prod;User
      ID=AATAPP;Pwd=XXXXXXXXXX;Initial Catalog=AAT;" />
</connectionStrings>
...

```

Test database connections string

Production database connection string

Figure 64: Configuration File – Connection Strings.

Based on the architecture, the authentication variables need to be set. The application will impersonate the user, so the application does not have to pass the user credentials. The authentication string information is shown in Figure 65.

```

...
<system.web>
  ...
  <!--
    The <authentication> section enables configuration
    of the security authentication mode used by
    ASP.NET to identify an incoming user.
  -->
  <authentication mode="Windows" />
  <identity impersonate="true" />
  ...
</system.web>
...
</configuration>

```

Authentication mode is Windows and identity impersonate must be true

Figure 65: Configuration File – Authentication Settings.

Scheduled Tasks

There are occasions that there will be some processing done outside the normal processing in the application. In the case of the example application, periodic e-mail notifications are generated. On those occasions where processing outside the application is necessary, a Windows Service is the best way to handle that type of processing. The Windows Service executes on a set timeframe, processes and waits for the next execution.

Reporting

Reporting is a very important aspect of any business application, as it is important to be able to get information back out of the application. There are reporting applications that can be implemented to gain reporting functionality, which would have to be built into an application. In the case of the example application, reporting is handled in an external reporting application where scheduling, security and a single point of reporting can be leveraged.

Chapter 5 – Project History

The example application, Audit Action Tracker, was designed and developed over a four month period of time. The project was broken down into five phases: documentation, design, construction, testing and implementation.

The documentation phase consisted of working with the users to formulate the initial application design, documenting that initial design, getting the user approval and developing the project plan. The documentation consisted of a combined scope and design document, as well as, the approval to proceed with the project.

The design phase consisted of working with the users to refine the initial design into a detailed design. Use cases, screen mockups and data elements were created and refined to a point where the developer and users could agree upon navigation and content of the application.

The construction phase consisted of each element of the application being built. Database elements were constructed first, tables, triggers, functions and stored procedures, as well as, the data migration plan based on the detailed design. Next the model elements were constructed; the domain, service and business layers were built based on the detailed design. Lastly, the user interface elements, the view and controller pieces were constructed.

The testing phase allowed the users to test the navigation and the particular functions to ensure they worked correctly. This phase also gave the users a chance to make minor changes to the system to better suit their needs. The developer in the testing phase had the opportunity to take the testing results and the change requests and make the necessary modifications to the application to better satisfy the needs of the users.

Table 5: Project history timeline. The table below contains a list of the phases and tasks and the periods of time that were taken to complete each task.

Phase	Task	End Date	End Date
Documentation	Create Project Short Form	7/2/2009	7/13/2009
	Create Project Plan	7/13/2009	7/13/2009
	Documentation Sign-off	7/13/2009	7/13/2009
Design	Design SQL Database	7/20/2009	7/27/2009
	Design Legacy Data Migration	7/28/2009	7/29/2009
	Design ASP.NET Web Pages	7/30/2009	8/7/2009
	Design Crystal XI Reports	8/10/2009	8/13/2009
	Design Complete	8/13/2009	8/13/2009
Construction	Construct SQL Database	8/20/2009	8/21/2009
	Construct Legacy Data Migration	8/24/2009	8/25/2009
	Construct ASP.NET Web Pages	8/26/2009	8/31/2009
	Construct Crystal XI Reports	9/1/2009	9/2/2009
	Construction Complete	9/2/2009	9/2/2009
Testing	Test Legacy Data Migration	9/9/2009	9/9/2009
	Test Admin Functions	9/10/2009	9/17/2009
	Test User Functions	9/18/2009	9/25/2009
	Test Database Functions	9/28/2009	10/5/2009
	Test Notifications	10/6/2009	10/7/2009
	Rework Discrepancies Found	10/8/2009	10/21/2009
	Testing Sign-off	10/21/2009	10/21/2009
Implementation	Implement SQL Database	11/2/2009	11/2/2009
	Implement Legacy Data Migration	11/2/2009	11/2/2009
	Implement ASP.NET Web Pages	11/2/2009	11/2/2009
	Implement Crystal XI Reports	11/2/2009	11/2/2009
	Implementation Sign-off	11/2/2009	11/2/2009

The implementation phase consisted of the actual implementation of the new application into the production environment. Putting all the pieces together, building the database, conducting the data migration and publishing the web application to the production web server.

Table 5 shows a breakdown by phase and task the project timeline.

Chapter 6 – Conclusions

The most important set of best practices for developing database driven, internal web-based business applications are principals, the principals of simplicity, consistency and user interaction. These principals drive the nuts and bolts of developing the application. In developing the example application, these principals presented themselves time after time.

Simplicity – finding the most concise way to do something, to make it easy to understand and use. In business applications, simple is just better. There is no need to add elements that are not necessary, just give the users what they need to satisfy their requirements and little more.

Consistency – doing the same task the same way each time, building the same function the same way each time and making the application look the same from function to function. The user will see consistency in the user interface, but the most important aspect of application consistency is in the construction that the user will never see. There should be a common theme, common naming convention and a common design philosophy. This commonality or consistency will help in the maintenance of the application and add additional functionality, as well as, helping other developers understand how the application is architected.

User interaction – giving users a voice in the development, they know what they need in a business application and they are going to have to ultimately use the application as a tool to make their work easier. Without the users and their requirements, there would be no need for the developer. Building a relationship between the user and developer is important; any development is a team effort.

Now for the nuts and bolts; exactly how to develop a database-driven, internal web-based business application from the ground up. The practices, themselves, employ simplicity, consistency and user interaction.

The initial design is very important, get the users involved early in the initial design phase as their knowledge is extremely important and they know what they need and want in an application. Communication is paramount, meetings, visual aids and documentation. Constantly working together with the key group; ensuring that everyone understands the design and has a chance to voice their opinions. The point of this phase is to refine the application on paper before even one line of code is written. Throughout the process the users must validate the ongoing work, since they should be able to identify any issues or rethink functionality before major rework would be required. The users must be able to test in the same way as they would use the application to identify workflow issues.

In a database driven application, the database is the basis for the design. All elements flow from the database, tables should equate to objects and stored procedures should equate to service implementations. The database should take advantage of self incrementing record keys, primary and foreign keys to preserve data integrity, simple logging techniques for audit purposes and simplify and reuse database objects where possible.

The use of design patterns in the application allows the development to be done more quickly and keep overall consistency in the architecture, which translates to maintainability and scalability in the future. The Model View Controller (MVC) design pattern takes a small step away from simplicity, but makes up for it in consistency. Understanding that, the developer should use it in a limited form. There are clear advantages to the single point of integration that the Factory and Layer Supertype patterns give the developer. The Separated Interface and Plugin

patterns allow for hiding technology and the ability to swap that technology out without having to rewrite the application.

Once again, the principals of simplicity and consistency are very important in the user interface. Presenting a simple consistent look and feel to the application is very important for the user interaction, as well as, the maintenance of the application. Clutter free windows, simple popup windows and controls, in addition to the ability to globally change the look and feel of the application, make the user experience more favorable. Presenting field validation messages before any processing takes place is very helpful to the user. Clearly displaying processing messages; both successful and understandable failure messages, assists the user in understanding the processing happening behind the scenes in the application.

Therefore, the best practices for designing and implementing a database driven, internal web-based business application are themselves simple. Get the users involved throughout the process, design the database as the basis to the rest of the application, build the application with commonality and maintainability in mind and, lastly, make the user interface clear and easy to use, that is the ultimate goal.

References

- Armstrong, D. J. (2006, Feb.). The quarks of object-oriented development. *Communications of the ACM*, 49(2), 123–128. doi:10.1145/1113034.1113040.
- Chen, A. N. K., Goes, P.B., Gupta, A., & Marsden, J. R. (2004, June). Database design in the modern organization—identifying robust structures under changing query patterns and arrival rate conditions. *Decision Support Systems*, 37(3), 435-447.
- Davidson, L. (2007, Feb. 26). *Ten common database design mistakes*. Retrieved from <http://www.simple-talk.com/sql/database-administration/ten-common-database-design-mistakes/>
- Fowler, M. (2003). Patterns. *IEEE Software*, 20(2), 56-57. doi:10.1109/MS.2003.1184168.
- Fowler, M., Sadalage, P. (2003, Jan.). *Evolutionary database design*. Retrieved from <http://www.martinfowler.com/articles/evodb.html>
- Fraternali, P. (1999, Sept.). Tools and approaches for developing data-intensive web applications: a survey. *ACM Computing Surveys (CSUR) archive*. 31(3), 227-263. doi:10.1145/331499.331502.
- Hager, D., Kibler, C., & Zack, L. (1999, May). The basics of user-friendly web design. *Journal for Quality & Participation*, 22(3), 58-61. Retrieved from Academic Search Premier database.
- Hice, R. (2008, November). Surrounded: The web is inescapable. *Scientific Computing*, 25(6), 18-20. Retrieved from Academic Search Premier database.
- Kotek, B. (2002, Oct. 30). *MVC design pattern brings about better organization and code reuse*. Retrieved from http://articles.techrepublic.com.com/5100-10878_11-1049862.html

Meyers, S. (2004). The most important design guideline? *IEEE Software*, 21(4), 14-16.

doi:10.1109/MS.2004.29.

Pattern. (2009). Retrieved from <http://dictionary.reference.com/browse/pattern>

Appendix A

Design Document

Project Name: **Audit Action Tracker (AAT)**

Author Stephen C. Rash
Date July 13, 2009

Revision & Sign-off Sheet

Change Record

Date	Author	Version	Change Reference
7/2/2009	Stephen C. Rash	1.0	Initial Document Development
7/6/2009	Stephen C. Rash	1.1	Updates
7/9/2009	Stephen C. Rash	1.2	App Name Change / Updates
7/13/2009	Stephen C. Rash	1.3	Finalize

Reviewers

Name	Position	Date	Approval
John Doe	Manager-EHS	7/15/2009	

Estimated Hours

Estimated hours for this project are between 250-300 hours.

Objective & Scope

The current Action Tracking System (ATS) functionality is outdated, cumbersome and time consuming for the users. ATS lacks the functionality and scalability required by the users to perform their job. The users want a system that is simpler and more streamlined which facilitates quick and easy user interaction, has enhanced security features and has better report generation features.

The objective of this project is to design a replacement application for ATS, which consists of three separate Visual Basic 6 applications, ATS, ATSUpload and ATSAuto. The new Audit Action Tracker (AAT) application will make the user interface simpler for the end user by taking advantage of Web-based (ASP.NET) technology, redesign the storage of data (MSSQL 2005 database) using tables, triggers, procedures and views to better manage data and develop reports (Crystal Reports XI) for display in our company wide reporting system (BusinessObjects Enterprise XI). This system will meet the functional and security requirements by managing the data, capturing an audit trail, and making the data more accessible and reportable.

Functional Requirements

The AAT application will contain the following functionality:

1. Security
 - a. Based on the user logging in, the system will search the Corporate Directory to find the user's structure and determine what data the user will be able to view and to what level of access (Add, Change, Read-Only) the user will have.
 - b. Also based on the user logging in, the system will determine if the user falls into the Admin or a TeamLead groups to allow additional system functions.
 - c. There will be 4 types of user access:
 - i. Admin – System Administrators (NT Group).
 - ii. TeamLead – Audit Team Leaders (NT Groups by Functional Group).
 - iii. Assignees – Individual responsible for the Action Item (Action Item Record).
 - iv. ReadOnly – The Location Managers and Supervisors and Manages above the Location Manager and Action Item Assignees.
2. Locations
 - a. Location information will be housed in a database table.
 - b. Location name, state and city information will be entered.
 - c. Location will be associated with a Business Unit, Region and Division.
 - d. System audit information will be housed in a database table and generated by triggers on the Location table.
3. Audits
 - a. Audit information will be housed in a database table.
 - b. Audit name, audit start and end dates will be entered.
 - c. Audit will be associated to a Location.
 - d. Facility Manager will be selected.
 - e. Audit Team Leader will be captured by login credentials.
 - f. Audit Team Members will be selected.
 - g. Audit Team Leader will complete Audit records.
 - h. System audit information will be housed in a database table and generated by triggers on the Audit table.
4. Action Items
 - a. Action Item information will be housed in a database table.
 - b. Audit Team Leader will Upload or manually add/change Action Item records.
 - c. Action Item findings, references, due date and Assignee will be entered.
 - d. Assignee will change/complete Action Item records.
 - e. System notification to Assignee when added to an Action Item.
 - f. System audit information will be housed in a database table and generated by triggers on the Action Item table.

5. System Notifications

- a. Generate periodic notifications to Assignee, Assignee’s Supervisor and Assignee’s Supervisor’s Supervisor

Detailed Design

The design of the AAT application will consist of a database (tables, triggers, procedures and views), Web-based users interface (ASP.NET with VB code behind) and reports (Crystal Reports XI accessed via BusinessObjects Enterprise):

Audit Maintenance



Audit Maintenance

Location

Complete

Search

#	Name	Location	Start Date	End Date	Comments
(1)					

Save |
 Reset |
 Complete

*Audit Type

*Audit

*Location (4)

*Start Date (5)

*End Date (5)

*Audit Lead (6)

*Location Manager (6)

Comments

- (1) Audit Search Grid User Control
 - A data grid to display the Audits for a particular Audit Lead or user in the Corporate Directory Hierarchy
 - Should display audit_name, audit_loc_name, audit_start_date, audit_end_date and audit_complete
 - Selected item should open and populate Audit Maintenance by audit_id
 - Data Grid only visible if Current User is in Audit Lead or Audit Admin

- (4) Location Select User Control
 - A User Control to search for and select a single Location

- (5) Date Select User Control
 - A User Control to display a calendar and select a date (Required)

- (6) User Select User Control
 - A User Control to search for and select a single User (user id) from the Corporate Directory

Audit Team



Current User
V1.0.0

- (9) Audit Team Grid
 - Display users of the Audit Team based on audit_id
 - Audit Team Member can be deleted from grid

- (6) User Select User Control
 - A User Control to search for and select a single User (user id) from the Corporate Directory

Action Item Upload



Action Item Upload

*File

Current User
V1.0.0

Action Item Maintenance



Action Item Maintenance

Priority Complete

Search

					(2)

*Assigned To (6)

*Priority

*Due Date (5)

*Finding

*Recommended Action

*Corrective Action

*Regulation

*Reference Type

*Reference

(2) Action Item Grid User Control

- A data grid to display the Action Items for a particular Audit Lead / Assignee or user in the Corporate Directory Hierarchy.
- Should display item_name, item_priority, item_due_date and item_complete
- Selected item should open and populate Action Item Maintenance by item_id.

(5) Date Select User Control

- A User Control to display a calendar and select a date (Required)

(6) User Select User Control

- A User Control to search for and select a single User (user id) from the Corporate Directory

Code Maintenance



Code Maintenance

Code Type

Code

Search

Code	Code Name	Code Type	Code Active	Code Dependency

(7)

|

*Code Type

*Code

Dependency

Code Active

Current User
V1.0.0

(7) Code Search Grid

- A grid to search for and select codes
- Should display code_type_id, code_name and code_active
- Selected item should open and populate Code Maintenance by code_id.

Location Maintenance



Location Maintenance

Location

State / Province

🔍 Search (8)

📁 Save | 🔄 Reset

*Location

*Business Unit

*Region

*Division

*Country

*Address 1

Address 2

Address 3

*City

*State / Province

*Postal Code

Location Active

Current User
V1.0.0

(8) Location Grid

- A grid to search for and select locations
- Should display loc_name, loc_city, loc_zip and loc_active
- Selected item should open and populate Code Maintenance by loc_id.

Audit Table Maintenance

Audit Table Maintenance

From Audit Lead 🔍 (6)

To Audit Lead 🔍 (6)

Change

From Manager 🔍 (6)

To Manager 🔍 (6)

Change

Current User
V1.0.0

Audit Team Lead User Update (From User -> To User)

(6) User Select User Control (From)

- A User Control to search for and select a single User (user id) from the Corporate Directory

(6) User Select User Control (To)

- A User Control to search for and select a single User (user id) from the Corporate Directory

Audit Location Manager User Update (From User -> To User)

(6) User Select User Control (From)

- A User Control to search for and select a single User (user id) from the Corporate Directory

(6) User Select User Control (To)

- A User Control to search for and select a single User (user id) from the Corporate Directory

Action Item Table Maintenance

Action Item Table Maintenance

From Assignee (6)

To Assignee (6)

Change

Current User
V1.0.0

Action Item Assignee User Update (From User -> To User)

(6) User Select User Control (From)

- A User Control to search for and select a single User (user id) from the Corporate Directory

(6) User Select User Control (To)

- A User Control to search for and select a single User (user id) from the Corporate Directory

Database (Tables)

Name	Description
tbl_code_type	Holds code type information, what code relates to what list for use in the system
tbl_code_audit	Holds code audit data, who, what and when the code record was inserted, changed or deleted by use of Triggers
tbl_code	Holds code specific information
tbl_location_audit	Holds location audit data, who, what and when the location record was inserted, changed or deleted by use of Triggers
tbl_location	Holds location data, the specific location where the audit is preformed
tbl_audit_audit	Holds 'audit' audit data, who, what and when the audit record was inserted, changed or deleted by use of Triggers
tbl_audit	Holds audit data, audit specific information, the where the audit was preformed, who preformed it and who is the responsible manager
tbl_audit_team_audit	Holds audit team audit data, who, what and when the audit team record was inserted, changed or deleted by use of Triggers
tbl_audit_team	Holds audit team data, what person(s) conducted the audit.
tbl_action_item_load	Holds action item upload data, temporary load information to be verified and loaded into the action item table
tbl_action_item_audit	Holds action item audit data, who, what and when the action item record was inserted, changed or deleted by use of Triggers
tbl_action_item	Holds action item data, event header detail lines... the type of waste to dispose of

Database (Triggers)

Name	Description
tgr_code_ins	Logs inserts to the code table
tgr_code_upd	Logs updates to the code table
tgr_code_del	Logs deletes from the code table
tgr_location_ins	Logs inserts to the location table
tgr_location_upd	Logs updates to the location table
tgr_location_del	Logs deletes from the location table
tgr_audit_ins	Logs inserts to the audit table
tgr_audit_upd	Logs updates to the audit table
tgr_audit_del	Logs deletes from the audit table
tgr_audit_team_ins	Logs inserts to the audit team table
tgr_audit_team_upd	Logs updates to the audit team table
tgr_audit_team_del	Logs deletes from the audit team table
tgr_action_item_ins	Logs inserts to the action item table
tgr_action_item_upd	Logs updates to the action item table
tgr_action_item_del	Logs deletes from the action item table

Database (Procedures)

Name	Description
sp_code_type_sel	Selects code type table record information
sp_code_sel	Selects code table record information
sp_code_ins	Inserts code table record information
sp_code_upd	Updates code table record information
sp_location_sel	Selects location table record information
sp_location_ins	Inserts location table record information
sp_location_upd	Updates location table record information
sp_audit_opn	Sets the Status to 'OPEN' on audit table record information
sp_audit_upd	Updates certain fields on audit table record information
sp_action_item_opn	Sets the Status to 'OPEN' on action item table record information
sp_action_item_upd	Updates certain fields on action item table record information
sp_audit_sel	Selects audit table record information
sp_audit_ins	Inserts audit table record information
sp_audit_upd	Updates audit table record information
sp_corp_dir_sel	Selects Corporate Directory record information
sp_audit_team_sel	Selects audit team table record information
sp_audit_team_ins	Inserts audit team table record information
sp_audit_team_upd	Updates audit team table record information
sp_audit_team_del	Deletes audit team table record information
sp_action_item_load_sel	Selects action item load table record information
sp_action_item_load_ins	Inserts action item load table record information
sp_action_item_load_upd	Updates action item load table record information
sp_action_item_load_del	Deletes action item load table record information
sp_action_item_load_xfer	Selects action item load table record information
sp_action_item_sel	Selects action item table record information
sp_action_item_ins	Inserts action item table record information
sp_action_item_upd	Updates action item table record information

Database (Views)

Name	Description
vw_all_audit_data	Selects All types of Audits data for reporting
vw_audit_audit_data	Selects Audit type of Audit data for reporting
vw_assessment_audit_data	Selects Assessment of Audit type data for reporting
vw_security_audit_data	Selects Security type of Audit data for reporting
vw_psm_audit_data	Selects PSM type of Audit data for reporting
vw_all_action_data	Selects All types of Action Items data for reporting
vw_audit_action_data	Selects Audit type of Action Item data for reporting
vw_assessment_action_data	Selects Assessment type of Action Item data for reporting
vw_security_action_data	Selects Security type of Action Item data for reporting
vw_psm_action_data	Selects PSM type of Action Item data for reporting

Database (Groups)

Name	Description
AAT_SysAdmin	System Administrators
AAT_AuditLead	Audit Team Leaders
AAT_AssmtLead	Assessment Team Leaders
AAT_SecLead	Security Team Leaders
AAT_PSMLead	PSM Team Leaders

Reports

Name	Parameters
All Detail Data	Date Range, Business Unit, Region, Division, Open/Completed
Audit Detail Data	Date Range, Business Unit, Region, Division, Open/Completed
Assessment Detail Data	Date Range, Business Unit, Region, Division, Open/Completed
Security Detail Data	Date Range, Business Unit, Region, Division, Open/Completed
PSM Detail Data	Date Range, Business Unit, Region, Division, Open/Completed
All Detail Data	Due Date Range, Priority, Open/Completed
Audit Detail Data	Due Date Range, Priority, Open/Completed
Assessment Detail Data	Due Date Range, Priority, Open/Completed
Security Detail Data	Due Date Range, Priority, Open/Completed
PSM Detail Data	Due Date Range, Priority, Open/Completed

Testing Scenarios

Test the following web-based AAT application functionality:

1. Code Maintenance
 - a. Search for an existing record
 - i. Code Search page opens
 - ii. Parameters limit search
 - iii. Search returns results
 - iv. Returns resulting record to the Code Maintenance page
 - b. Add a record
 - i. Required fields must be filled in to Save
 - ii. Inserts entire record into the database
 - iii. Inserts 'ADD' record into the Code Audit table – Admin
 - c. Change a record
 - i. Required fields must be filled in to Save
 - ii. Updates correct record into the database
 - iii. Inserts 'CHG' record into the Code Audit table – Admin

2. Location Maintenance
 - a. Search for an existing record
 - i. Location Search page opens
 - ii. Parameters limit search
 - iii. Search returns results
 - iv. Returns resulting record to the Location Maintenance page
 - b. Add a record
 - i. Required fields must be filled in to Save
 - ii. Inserts entire record into the database
 - iii. Inserts 'ADD' record into the Location Audit table – Admin
 - c. Change a record
 - i. Required fields must be filled in to Save
 - ii. Updates correct record into the database
 - iii. Inserts 'CHG' record into the Location Audit table – Admin

3. Audit Table Maintenance
 - a. Re-open Audit Records
 - i. Generates Notification that the Audit record was Changed (See #9)
 - ii. Status field on the Audit record is set to 'OPEN'
 - iii. Inserts 'CHG' record into the Audit Audit table – Admin
 - b. Update Audit Records
 - i. Generates Notification that the Audit record was Changed (See #9)
 - ii. Field data matching Criteria is changed
 - iii. Inserts 'CHG' record into the Audit Audit table – Admin
 - c. Delete Audit Records
 - i. Generates Notification that the Audit/Action Item records were Deleted (See #9)
 - ii. Deletes Audit and all associated Action Item records from the database

- iii. Inserts 'DEL' record into the Audit/Action Item Audit tables – Admin
4. Action Item Table Maintenance
- a. Re-open Action Item Records
 - i. Generates Notification that the Action Item record was Changed (See #9)
 - ii. Status field on the Action Item record is set to 'OPEN'
 - iii. Inserts 'CHG' record into the Action Item Audit table – Admin
 - b. Update Action Item Records
 - i. Generates Notification that the Action Item record was Changed (See #9)
 - ii. Field data matching Criteria is changed
 - iii. Inserts 'CHG' record into the Action Item Audit table – Admin
 - c. Delete Action Item Records
 - i. Generates Notification that the Action Item record was Deleted (See #9)
 - ii. Deletes Action Item record from the database
 - iii. Inserts 'DEL' record into the Action Item Audit table – Admin
5. Audit Maintenance
- a. Search for an existing record
 - i. Audit Search page opens
 - ii. Parameters limit search
 - iii. Search returns results
 - iv. Returns resulting record to the Audit Maintenance page
 - b. Add a record
 - i. Inserts entire record into the database
 - ii. Inserts 'ADD' record into the Audit Audit table – Admin
 - c. Change a record
 - i. Updates correct record into the database
 - ii. Inserts 'CHG' record into the Audit Audit table – Admin
 - d. Complete a record
 - i. Updates correct record into the database
 - ii. Inserts 'CHG' record into the Audit Audit table – Admin
 - e. Add Audit Team members (See #6)
 - f. Upload associated Action Item records (See #7)
6. Audit Team Maintenance (Add, Change and Delete Records)
- a. Add a record
 - i. Audit Team Add page opens
 - ii. Parameters limit search
 - iii. Search returns results
 - iv. Inserts entire record into the database
 - v. Inserts 'ADD' record into the Audit Team Audit table – Admin
 - b. Delete a record
 - i. Deletes correct record into the database

- ii. Inserts 'DEL' record into the Audit Team Audit table – Admin
7. Action Item Upload
- a. Upload a file
 - i. Browse for formatted MS Excel upload file
 - ii. Inserts all file contents into the Action Item Load table
 - b. Resolve any errors
 - i. Identify any error fields
 - ii. Update and save any error fields
 - c. Add all records
 - i. Inserts entire record into the database
 - ii. Inserts 'ADD' record into the Action Item Audit table – Admin
8. Action Item Maintenance
- a. Search for an existing record
 - i. Action Item Search page opens
 - ii. Parameters limit search
 - iii. Search returns results
 - iv. Returns resulting record to the Action Item Maintenance page
 - b. Add a record
 - i. Generates Notification that the Action Item record was Added (See #9)
 - ii. Inserts entire record into the database
 - iii. Inserts 'ADD' record into the Action Item Audit table – Admin
 - c. Change a record
 - i. Generates Notification that the Action Item record was Changed (See #9)
 - ii. Updates correct record into the database
 - iii. Inserts 'CHG' record into the Action Item Audit table – Admin
 - d. Complete a record
 - i. Generates Notification that the Action Item record was Completed (See #9)
 - ii. Updates correct record into the database
 - iii. Inserts 'CHG' record into the Action Item Audit table – Admin
9. Notifications
- a. Notification was generated
 - b. Notification was e-mailed to correct individuals
 - c. Notification was copied to the AAT mailbox

Annotated Bibliography

Armstrong, D. J. (2006, Feb.). The quarks of object-oriented development. *Communications of the ACM*, 49(2), 123–128. doi:10.1145/1113034.1113040.

The author took an in depth look at Object-Oriented Development (OO) as to why it has not lived up to its potential. The author asserts that there are still issues with understanding the basic concepts of OO and how they fit into a coherent scheme. Armstrong outlined the quarks of OO by defining and giving the reader some background on the major concepts of OO; inheritance, object, class, encapsulation, method, message passing, polymorphism and abstraction. Armstrong then examined the OO taxonomy and how the concepts fit together to create an approach into two constructs; Structure (Abstraction, Class, Encapsulation, inheritance and Object) and Behavior (Message Passing, Method and Polymorphism). Structure is focuses on the relationships between the classes and objects and also how they are structured. Behavior focuses on the object actions within the system. The author then explains why there has been no consensus on the concepts of OO because there are no set of standards established to aid in the learning of OO. This was a very good article for a reader who was unsure of the concepts and structure of OO. The concepts were defined very well and how they fit together was also explained in such a way that would be understandable. The author was knowledgeable and seemed to understand how to explain the concepts to others.

Chen, A. N. K., Goes, P.B., Gupta, A., & Marsden, J. R. (2004, June). Database design in the modern organization—identifying robust structures under changing query patterns and arrival rate conditions. *Decision Support Systems*, 37(3), 435-447.

The authors illustrate that there are many variables to selecting the best database design to satisfy a specific need, there is no one solution that would fit under all conditions. The authors present their approach to understanding the best design for a given database, their approach consisted of five steps; construct a feasible database; measure processing times for each query type; identify top performers; evaluate the top performers with additional performance measures to identify robust performers; evaluate the robust performers across complexity levels to make selections. The authors laid out their example database application environment; the tables and how they relate as well as keys and data sizing. The example database testing was comprehensive and used a query pattern to evaluate 5 components on both non-congested and congested systems. The authors were able to evaluate and select potential good performers using their five steps to determine robust performers. This article was written at a high-level, it was understandable to someone who had little prior knowledge of the subject but was not very useful in understanding how to replicate the process.

Cook, R. (2007, June 19). *Securing the endpoints: The 10 most common internal security threats*. Retrieved June. 17, 2009, from the CIO.com web site: http://www.cio.com/article/120101/Securing_the_Endpoints_The_Most_Common_Internal_Security_Threats

The author looks at the top ten most common security threats to internal networks. The analysis was done based on endpoints; any device connected to the corporate network, desktops, laptops, PDAs and cell phones. The ten major problem areas are, USB Devices: anyone who can get access to a network asset, can download or upload from a USB drive and there is little security in place to stop that. Peer-to-Peer File Sharing: unauthorized programs allowing file sharing through a secure network. Antivirus Problems: companies not updating their antivirus software often and on a regular basis. Outdated Microsoft Service Packs: companies not keeping their vendor supplied software current. Missing Security Agents: security agents not being installed which can alert companies as to network traffic, missing company assets or verify that software patches have been installed. Unauthorized Remote-Control Software: software that can allow someone possibly outside the network to access and control an internal network asset. Media Files: unauthorized audio and video files can contain hidden malicious programs. Unnecessary Modems: an unsecured modem is a direct pathway into a company's network. Unauthorized or Unsecured Synchronization Software: software that synchronizes different devices can potentially transfer sensitive company data without the user even knowing it. Wireless Connectivity: most laptop computers have a built in wireless access, which could be used for malicious purposes. It is important to control as many of these security threats as you can, you will never be able to eliminate all of them, but you should strive to attain as close to that as you can. This was a very interesting and thought provoking article, it really opened my eyes to the security threats that are very commonly used.

Davidson, L. (2007, Feb. 26). *Ten common database design mistakes*. Retrieved June. 15, 2009, from the Red Gate Software web site: <http://www.simple-talk.com/sql/database-administration/ten-common-database-design-mistakes/>

The author outlines the ten most common mistakes in designing databases and gives examples and real world insight into the problem. Poor design/planning; the database is the cornerstone of most projects, so every aspect must be thought out before a line of code is written. Ignoring normalization; a single table cannot do it all, break the data down into as small a logical group as you can for performance and ease of development. Poor naming standards; consistency and readability are the keys, name it what it is and be consistent across the application. Lack of documentation; good standards are only part of it, document aspects so someone else can understand how the system works, it just might be you who needs a refresher. One table to hold all domain values; break them up into smaller logical groups, it is more difficult, but worth the time for maintainability. Using identity/guid columns as your only key; an identity field should be used in conjunction with a natural key, something a user could understand. Not using SQL facilities to protect data integrity; base rules such as nullability should be implemented in the database, any aspects that are rigid and will not change. Not using stored procedures to access data;

stored procedures insulate the database layer from the users and assist in maintainability, encapsulation, security and performance. Trying to build generic objects; be specific, there are performance concerns to trying to be too generic. Lack of testing; test the database piece by piece to ensure it is working, it is harder to troubleshoot and correct further down the line. This was a very well written article, full of real world examples from an author who is both passionate and knowledgeable on the subject.

Fowler, M. (2003). Patterns. *IEEE Software*, 20(2), 56-57. doi:10.1109/MS.2003.1184168.

Fowler states his reasons for using design patterns. Patterns are a good way to assist the designer in solving problems in a controlled manor, solving recurring problems with common solutions and designing in a consistent structured way. Patterns are a tool to assist in solving a problem; they themselves are not a solution. Implementing patterns in libraries is not advisable, the pattern may be hard to find and understand; developers move from language to language the pattern by itself would be more useful and the library can implement a pattern, but it is up the developer on how to use it. Experts might find patterns unnecessary, they might not learn anything new, but they can be a good tool to teach others and have a common vocabulary so everyone can understand with little explanation. Pattern overuse is a problem; if a pattern does not contribute it should be removed. The author has a great deal of experience in this field and his insights are displayed in this article. The article is a good piece to understand the important aspects of design patterns.

Fowler, M., Sadalage, P. (2003, Jan.). *Evolutionary database design*. Retrieved June 19, 2009, from the Martin Fowler web site: <http://www.martinfowler.com/articles/evodb.html>

The authors put forth some very interesting ideas about evolutionary database design. The first aspect was dealing with change; the design is an on-going process, is iterative in nature and the designer might run through many life-cycles over the life of the project. The authors also highlighted the fact that they not solved all the problems of evolutionary databases. This approach involves several practices, DBAs collaborate closely with developers; constant communication is very important. Everybody gets their own database instance; developers get their own sandbox to play in that will not affect anyone else. Developers frequently integrate into a shared master; development work flows frequently to the master from which all work flows back down. A database consists of schema and test data; the actual database and standardized test data so all developers test with the same subset of data. All changes are database refactorings; control the changes, change all aspects so nothing becomes disconnected. Automate the refactorings; script all changes so they can be consistently applied. Automatically update all database developers; push the changes from the master to the developers automatically so everyone has the same database to develop on and no developer is disconnected from the others. Clearly separate all database access code; have a clearly defined data access layer in the application, invisible to changes in the actual database. The authors also highlighted variations to this design, keeping multiple database lineages; in more complex applications multiple versions of the database may need to be maintained. You don't need a DBA; most of the work can be done by developers. The authors also stated it

is important to automate as much of the repetitive tasks as can be. This was a very interesting article; it presented a new way of looking at database design and outlined best practices for that type of development.

Fraternali, P. (1999, Sept.). Tools and approaches for developing data-intensive web applications: a survey. *ACM Computing Surveys (CSUR) archive*. 31(3), 227-263. doi:10.1145/331499.331502.

The author outlined web-application development in terms of software engineering, architectural and applicative issues. Process: the development lifecycle of the application, consisting of the following steps: requirements analysis, conceptualization, prototyping and validation, design, implementation and finally evolution and maintenance. Models, Languages, and Notation: characterized by three major design dimensions: structure, navigation and presentation. Reuse: the ability to reuse an object at any level in the development cycle. Architecture: the physical arrangement of the application and its access. Usability: the presentation and navigation as well as the flexibility and proactive nature of the application. The author also outlined the current development tools. Visual Editors and Site Managers: a visual way to write the underlying web code. Web-enabled Hypermedia Authoring Tools: similar to visual editors, but from a different origin for developing off-line code. Web-DBPL Integrators: database driven development tools. Web Form Editors, Report Writers, and Database Publishing Wizards: using traditional database design concepts and development tools to create data-intensive applications Web applications. Multiparadigm Tools: a combination of the previously mentioned visual and database driven tools. Model-Driven Web Generators: use conceptual modeling and code generation techniques to the development of Web applications. Middleware, Search Engines, and Groupware: middleware is the communication piece between the web application and the database, search engines are logical navigation of the application and groupware provide access, collaboration and workflow. The author then evaluated the relationship between what was termed as “state-of-the-practice solutions” and relevant areas along with the research prospective. Fraternali also discussed in detail five research projects in data-intensive Web development. The author then discussed his background research in the areas of modeling notation, processes and other design tools. This was a very good article, there was a considerable amount of pertinent information as well as referential and background to the study. The research was comprehensive and the author’s conclusions were sound and well formulated.

Hager, D., Kibler, C., & Zack, L. (1999, May). The basics of user-friendly web design. *Journal for Quality & Participation*, 22(3), 58-61. Retrieved June 20, 2009, from Academic Search Premier database.

The authors discuss the challenges and techniques around creating Web applications in a user-centered approach. The advantages to Web applications also cause some problems; multiple browser compatibility, network connectivity and individual user browser customizations. The users must be involved in the design, without that involvement the application may be frustrating and not useful for users. Setting goals as to when the application is complete and can move into production with the understanding that it is not

perfect, but through feedback the application will improve. The designer must also know who they are designing the application for; what they should know, what their experiences have been, what they do in their job, what they expect from the application and what other applications have they used that may be helpful. Once the user has been understood, the actual tasks the application will perform are analyzed. With the task information the process can start; build a prototype and work with the users, research how others solved similar issues, walkthrough the design with the users to get feedback, build the applications and allow a subset of users to test it and finally, distribute the application to the entire population and survey them for feedback. This was a very good article; the authors knew their subject matter and presented it well. I found some useful tips on web application design.

Hice, R. (2008, November). Surrounded: The web is inescapable. *Scientific Computing*, 25(6), 18-20. Retrieved June 20, 2009, from Academic Search Premier database.

The author started out with an amusing anecdote to illustrate how users are constantly connected to others by the cell phone. Hice continues on to explain through the use of cell phones and internet access on commercial airlines how more and more applications are becoming Web-based or Web-enabled. The author highlights how applications are migrating from PC or client/server based to Web-based. Companies started looking at centralizing applications using Citrix Metaframes to make them Web-available; the application was just running on a remote computer. Early attempts at Web-enabled applications meaning they still required software to be loaded on the workstation and server were written in HyperText Markup Language (HTML); they were just not as good a user interface as the applications they were replacing. More recently with the advent of eXtensible Markup Language (XML) and Web services the applications are truly becoming Web-based; better functionality and usability as a user interface. A good thought provoking article highlighting the trends of applications moving from PC or client/server based to fully Web-based.

Kotek, B. (2002, Oct. 30). *MVC design pattern brings about better organization and code reuse*. Retrieved June 16, 2009, from the TechRepublic web site:
http://articles.techrepublic.com.com/5100-10878_11-1049862.html

The author explains how MVC works to by enforcing the separation of the different aspects of the application into; the model, the view and the controller with each handling a different set of tasks. The view does very little processing, it just handles the input from the users and returns the output. The controller interprets requests from the view and routes them to the appropriate portion of the model to complete the request. The model is the business logic and communication to the data storage which returns natural data to the controller and on to the view. The author also explains why MVC is an important design pattern for web applications. Multiple views can access a single model, because the view and model are disconnected, the views can be swapped out with no changes to the model. Changes to data access and business rules can be made easier within the model and changes there will be invisible to the controller and the view. The concept of a controller is also powerful, it connects the two independent pieces together, so either one can

change without affecting the other, it allows for reusability of the different pieces in the model and view. The author highlighted the drawbacks of the MVC pattern. MVC is complex and requires a great deal of planning and attention to detail. MVC might not be worth the trouble for small or even medium sized applications. This article was a good overview of the MVC design pattern. The author spoke to the subject with knowledge and understanding. I however disagree with the assertion that MVC is too much trouble for small or medium applications, if you understand the implementation, the advantages of the highly separated system outweigh the extra work in implementing MVC.

Meyers, S. (2004). The most important design guideline? *IEEE Software*, 21(4), 14-16.
doi:10.1109/MS.2004.29.

In this article the author emphasized many good practices for designing and developing good user interfaces. His underlying idea is to “make interfaces easy to use correctly and hard to use incorrectly.” Meyers states that it is the responsibility of the designer to make the interface user friendly and if they do not, it is their fault if anything goes wrong, not the user. The designer must design the interface to not allow the user to make mistakes. The author asserts that using drop-down lists to only allow the user to select valid values, but this is not always the ideal, it might cause more errors than it solves. The designer must consider all the ways a user could misuse the interface in considering a design. Another aspect to good design is releasing and destroying object no longer needed. Clean up will help with performance and keep the interface running smoothly. The author had a very good understanding of designing user interfaces. The article showed how important the actual design portion of development truly is, and that it is the ultimate responsibility of the designer to make the interface useable and perform well.