SUICIDE HOTLINE/RESPONSE TECHNICAL MANUAL

Version 1
May 2009

Department of Veterans Affairs
Office of Information and Technology
Medora Development Group

Revision History

Date	*Description*	*Author*
12/16/2008	Draft I	Jason Jones, Suicide Hotline Developer
1/27/2008	Draft II – added installation section, examples of how the Spring.NET Framework works.	Jason Jones, Suicide Hotline Developer
3/27/2009	Edited to describe new project structure in Visual Studio.NET	Jason Jones, Suicide Hotline Developer

Table of Contents

Introduction

The Suicide Hotline (SH) application provides enhanced tracking and reporting of the Suicide Hotline services provided to veterans by:

- The National Suicide Hotline at the Center for Excellence in Canandaigua
- Suicide Prevention Coordinators at nationwide VA Health Care Centers
- Health Technicians at the Center for Excellence in Canandaigua

Currently, there is no VistA software that meets the needs of the National Suicide Hotline. The current paper-based process, along with the VistA inter-facility consult system, does not provide comprehensive tracking, reporting, and monitoring capability.

The Suicide Hotline Version 1.0 application will provide entirely new functionality that will encompass and integrate all segments of the Suicide Hotline Services including Suicide Prevention Coordinators and Health Technicians roles.

The Suicide Hotline Technical Manual/Security Guide gives a technical description of the application and the security features. The intended audience of this guide is system administrators and developers supporting the Suicide Hotline product.

Benefits

- Eliminates time-intensive and inefficient paper reporting
- Complies with VistA Architecture
- Complies with 508 regulations, using W3C standards
- Accessible web-based application, via a web browser
- Supports the Office of Information Single Sign-on initiative
- User authentication via role based permissions
- User-friendly
- Seamless continuum of care
- Minimum user disruption
- Simplified data entry
- Better identification and treatment of veterans
- Consolidates data
- Enables Suicide Hotline call tracking and reporting capabilities
- Enables users to receive comprehensive views of a patient's suicide follow-up care across institutions
- Facilitates data tracking and auditing capabilities
- Improves accountability
- Enhanced reporting features
- Provides data standardization which improves and provides consolidated data reporting
- Enables research and provides outcomes tracking and reporting capabilities
- Improves VHA organizational communication

Enhanced Technology

- A single consolidated database and application will replace the current paper and inter-facility consult system
- Fulfills the congressional mandate for suicide hotline tracking and reporting
- Electronic referral process to track patient requests for service
- Progress Notes can be created through the application for assessments and clinical visits
- Nationwide centralization of Suicide Hotline services data to allow nationwide reporting
- Ad-hoc reporting capabilities
- Secure Web Access (128 Bit SSL) from any authorized VA workstation
- Improved technology using web browser access and improved data security, via the VHA intranet
- Uses modern system architecture which allows for faster system enhancements
- Enhancements will be rolled out to all users at the same time ensuring consistent data
- Patient lookup using the VistaWeb Multi-Site Lookup System
- Minimizes the maintenance and support required by IT support staff

VistA Software Requirements

No package installations or patches are needed to run Suicide Hotline.

Recommended Users

The intended audience for the Suicide Hotline Technical and Security Guide includes:

Information Resource Management (IRM)
Developers for the Office of Information and Technology (OI&T)

Related Manuals

National Suicide Hotline User Manual
Online Help is available from within the application

Documentation Retrieval

You can find the documentation files for Suicide Hotline on the Medora Sharepoint website. (http://medora.sharepoint.med.va.gov/sites/crisiscenter/default.aspx)

File Name	***Description***	***Retrieval Format***
TBD	Technical/Installation Manual	Binary
TBD	User Manual	Binary

The manuals will also be available on the VistA Documentation Library (VDL).

MDWS Web Services

Documentation for this product is available on the intranet at the following address: http://medora.sharepoint.med.va.gov/sites/MDWS.

This address takes you to the MDWS Project page, which has more detailed information regarding the software.

The link below allows access to the Suicide Hotline home pages:
http://medora.sharepoint.med.va.gov/sites/CrisisCenter

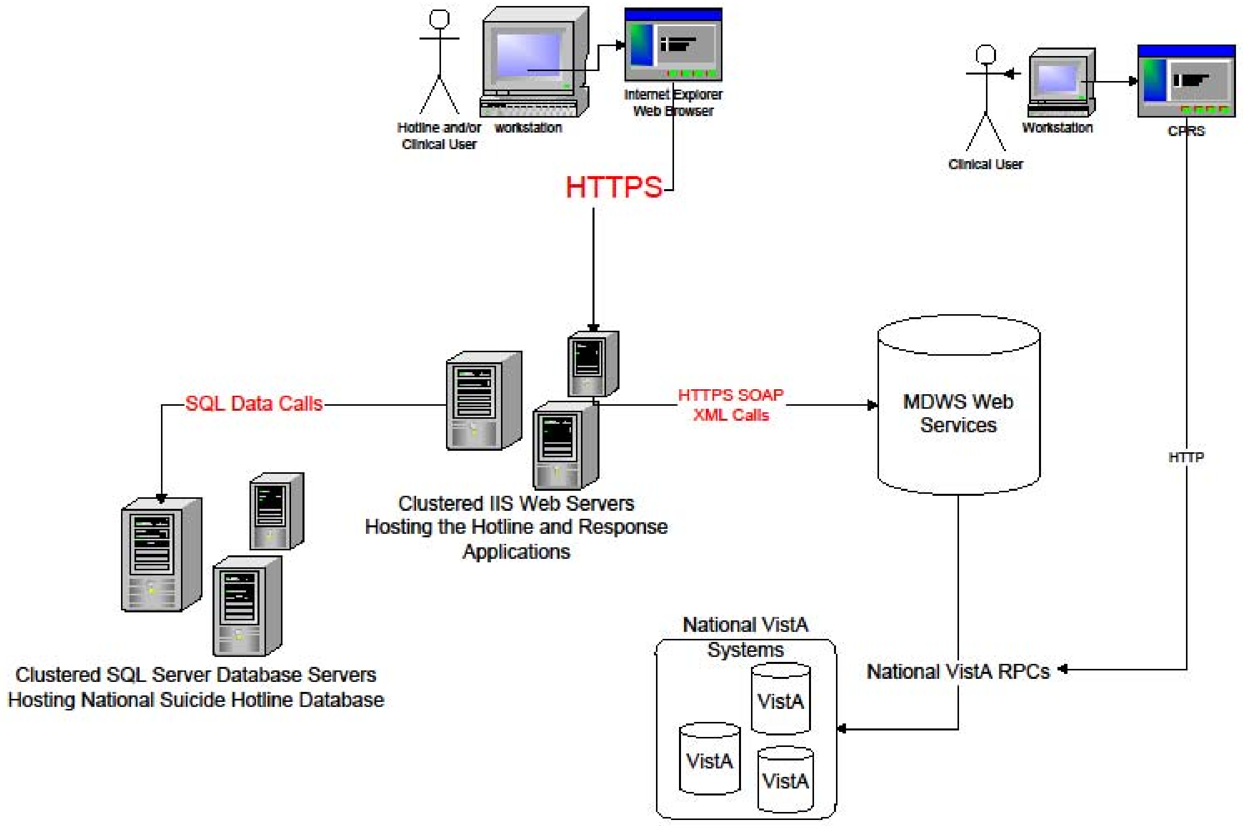
http://medora.sharepoint.med.va.gov/sites/CrisisCenterResponse

Implementation and Maintenance

Implementation is comprised of a centralized application:

Hosted in Austin Data Center.
User setup Suicide Hotline application

Suicide Hotline Configuration


Hotline and/or Clinical User
workstation
Internet Explorer Web Browser
HTTPS
Clinical User
Workstation
CPRS
SQL Data Calls
HTTPS SOAP XML Calls
MDWS Web Services
HTTP
Clustered IIS Web Servers Hosting the Hotline and Response Applications
Clustered SQL Server Database Servers Hosting National Suicide Hotline Database
National VistA Systems
VistA
VistA
VistA
National VistA RPCs

VistA Files

There are no files added to VistA with this software.

Security Keys

There are no VistA security keys used for this web application. However, the VistA Option Menu Item CPRS GUI CHART is required in order to use the VistA-related functions of the application, such as Login, Patient Lookup, and Progress Note writing.

Option Menu Item	***Description***
CPRS GUI CHART	All users of the system must have the CPRS GUI CHART Option.

Software Product Security

Suicide Hotline implements Single Sign-on (SSO) based on established VHA requirements. To fulfill these requirements, the Suicide Hotline uses MDWS VistA logon service. Users of the system can access Suicide Hotline by entering their current VistA Access and Verify Codes along with their institution. The user's information is checked against the VistA system, first by authenticating their access/ verify code, and institution combination. Additional authentication occurs inside the Suicide Hotline software to ensure the user has access to the Suicide Hotline application itself and to provide the user with a session containing their VistA user information as well as access role within the application. Access to the application is allowed to users with correct login credentials, who can only access the areas of the application to which their assigned role has access.

Patient and user context management via CCOW was not added to the application, as it was deemed unnecessary. In most cases, this application will be used by itself, not in conjunction with CPRS.

Secure socket layer (128Bit - SSL) encryption is used to protect data that is transmitted between the browser and the Web server.

Security Management

Data contained in and messaged to SH is compliant with all applicable organizational and legislative security and privacy policies: Health Insurance Portability and Accountability Act (HIPAA), the Privacy Act of 1996, Veterans Health Administration's Authentication, Authorization, Accountability (AAA) standards, guidance and procedures established by the VHA Health Information Security Office, and policy and guidance issued by the Department of Veterans Affairs Office of Cyber Security.

Application Security

Security risks to SH are mitigated by:

- C# Secure Coding Guidelines
- VistA Authentication through MDWS
- Role-based Access
- Audit Trail
- Network Security
- 128bit SSL (HTTPS) Access to Web Server
- SH Site Server Security
- Physical Security

Audit Trail

An audit trail creates retrievable records of the users' interactions with the system. The ability to trace users to edits of specific records creates an identification that can be accessed in the event of a security breach or system defect.

The audit trail starts when a user accesses, submits a new record, or updates an existing record. The user ID is captured from their login and is stored in the audit trail, along with the record modification details. The stored record can be used for reporting information about user transactions.

Remote Systems

This system connects to remote VistA systems throughout the organization. The data transmitted from a user's desktop to the Suicide Hotline web server is encrypted through 128bit SSL (HTTPS). Data transmitted from the application server to the database server is not encrypted.

Archiving/Purging

Archiving and purging capabilities are not currently available and were not requested or required.

Contingency Planning

Sites utilizing Suicide Hotline should develop a local contingency plan to use in the event of product problems in a live environment. The facility contingency plan must identify the procedure for maintaining functionality provided by this package in the event of system outage. Field station Information Security Officers (ISOs) may obtain assistance from their Regional Information Officer (RISO).

Electronic Signatures

N/A at this time.

File Security

No new VistA files are distributed with this product.

Official Policies

Suicide Hotline 1.0 software release references no official policy unique to the product regarding the modification of software and distribution of the version.

Suicide Hotline Application

Suicide Hotline is designed to run on a Windows application server running IIS version 6 or higher.

Web.config

This file exists in the root application directory per the ASP.NET 3.5 specifications. This file describes the configuration of the .NET container, and any other frameworks and modules being used, such as the Spring.NET application framework which is used in SH.

secret-DBConnections.xml

This file exists in the root application directory. It contains the setup information for the SQL Server database. This file is not contained in the Subversion Source Control code repository, for reasons of security. Anything in our source code repository can requested by anyone through the Freedom Of Information Act (FOIA). As such, we must keep system logins and passwords out of the source code so as not to create a potential security breach.

Bin Directory

The bin directory contains external .dll files needed for the SH application:

Library Name	***Description***
AjaxControlToolkit	Ajax Controls - used for Tabbed Layers
AJAXExtensionsToolbox.dll	Necessary for Ajax Controls
Spring.Web.dll	Spring.NET web framework
NHibernate.dll	NHibernate Object-Relational Mapper
Nullables.dll	Supports NHibernate
Spring.Core.dll	Core of the Spring.NET Framework
Spring.Data.dll	Spring Framework support
Spring.Services.dll	Spring Framework support
Spring.Data.NHibernate12.dll	Spring-NHibernate support classes
Spring.Aop.dll	Spring AOP support
System.Web.Extensions.dll	Expanded Web Controls for .NET
System.Web.Extensions.Design.dll	Dependency of System.Web.Extensions
Castle.DynamicProxy.dll	Dependency of the Spring-Hibernate dll.
log4net.dll	Log4Net logging
Common.Logging	Common Logging – works with Log4Net
Common.Logging.Log4Net.dll	Common Logging – works with Log4Net

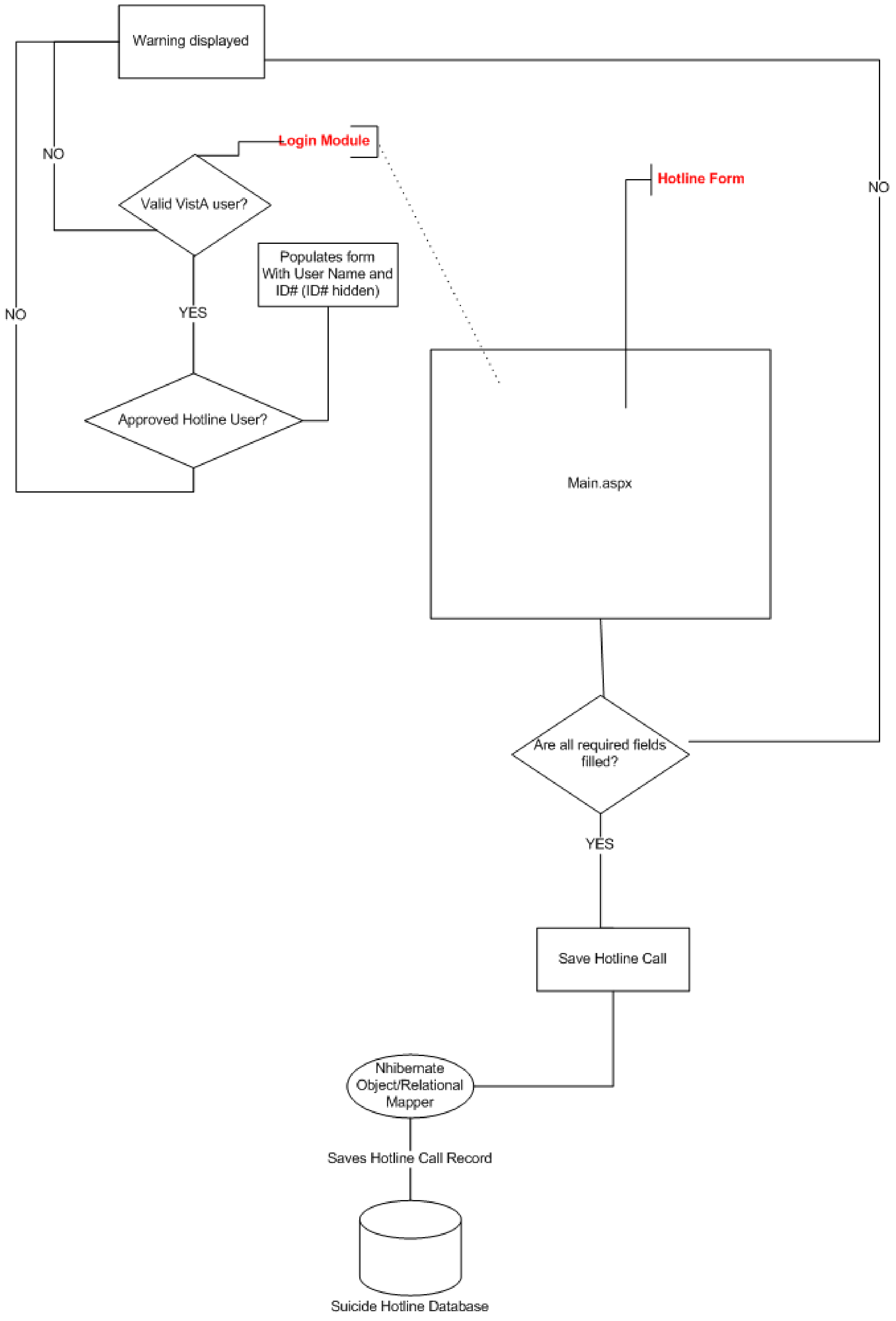
The Application Folder

The SH application folder contains the SH files needed to run the application. The folder contains a web directory and the bin directory.

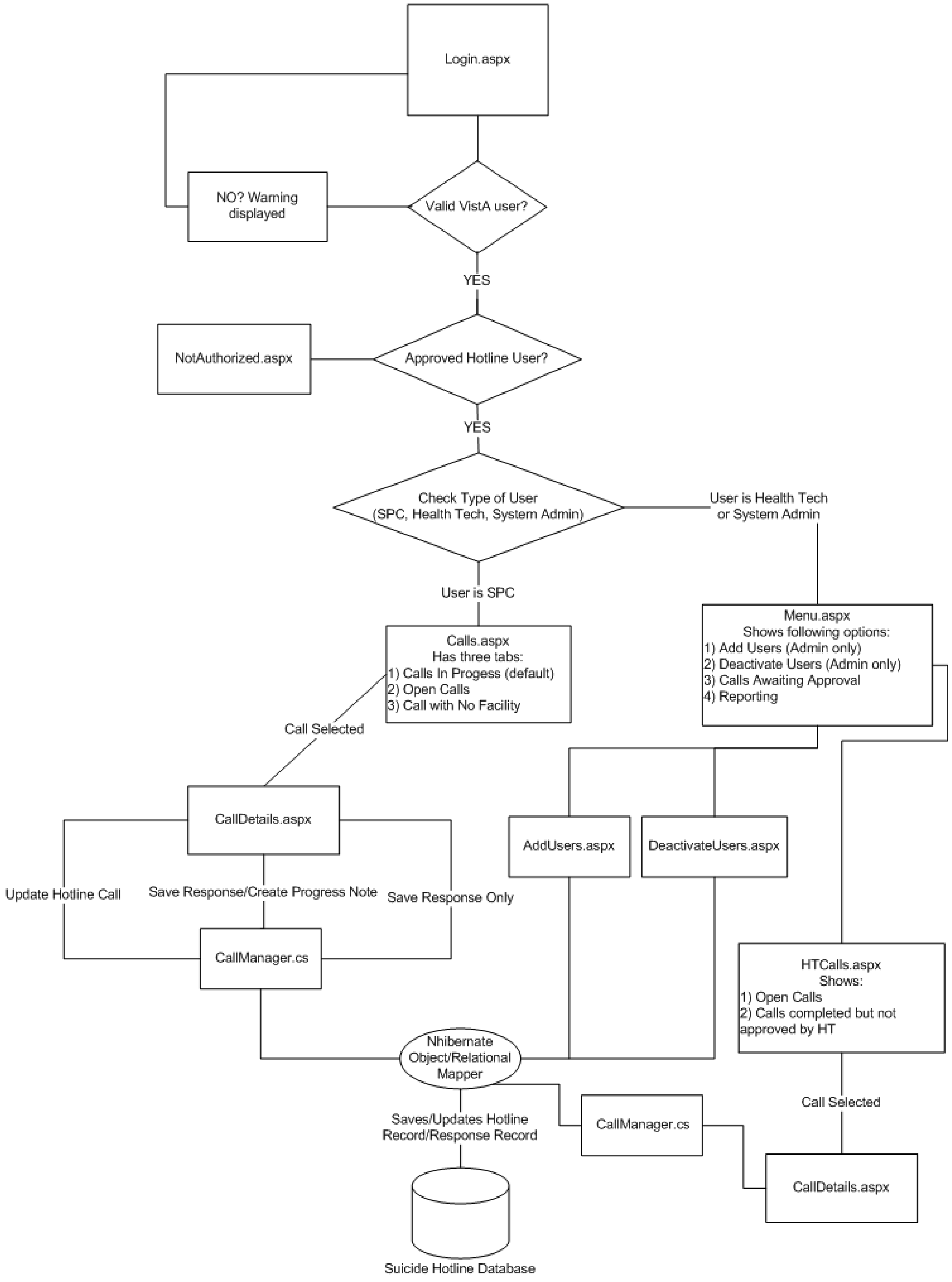
Directory Contents

Directory Name	*Description*
/	Root directory contains all web pages except reports and error handling
Reports	Contains all SH reports web pages (Response Module only)
CSS	Style Sheets
Images	Graphics for the site
ErrorHandling	Web pages for handling application errors and 404 (Page Not Found) errors
Scripts	Javascript files
Masters	“Master” template files – none are in use in this application.
Config	Configuration files for the Spring.NET Framework

Application Flow Diagram (Hotline Module)



Application Flow Diagram (Response Module)



Login.aspx
NO? Warning displayed
Valid VistA user?
YES
NotAuthorized.aspx
Approved Hotline User?
YES
Check Type of User (SPC, Health Tech, System Admin)
User is Health Tech or System Admin
User is SPC
Calls.aspx Has three tabs: 1) Calls In Progess (default) 2) Open Calls 3) Call with No Facility
Menu.aspx Shows following options: 1) Add Users (Admin only) 2) Deactivate Users (Admin only) 3) Calls Awaiting Approval 4) Reporting
Call Selected
CallDetails.aspx
AddUsers.aspx
DeactivateUsers.aspx
Update Hotline Call
Save Response/Create Progress Note
Save Response Only
CallManager.cs
HTCalls.aspx Shows: 1) Open Calls 2) Calls completed but not approved by HT
Nhibernate Object/Relational Mapper
Call Selected
CallManager.cs
Saves/Updates Hotline Record/Response Record
CallDetails.aspx
Suicide Hotline Database

The Spring.NET Framework

The Suicide Hotline application makes use of a concept that is gaining popularity in the programming world for its simplicity and facilitation of modular, non-interdependent, easily testable code. This concept is known as "Inversion of Control." The concept, in a nutshell, is to take the interdependencies among classes, and give the setup and control of those dependencies to an outside entity, which in this case is a framework. The idea is that if, say, one class needs to use an instance of another class, rather than hard-code or instantiate the class directly in the class, which then creates a "dependency" on that class (if the class changes or goes away, the class with its instantiation breaks), an outside framework with knowledge of both classes can "inject" an instance of the needed class into an instance of an interface of the needed class. This particular type of "Inversion of Control" (IoC) is called "Dependency Injection" (DI). The result of this method of handling dependencies is that a class can change considerably, but as long as it implements the interface called in the dependent class, the dependent classes will not be broken by the changes. In addition, it allows the programmer to test classes without their dependencies even being completed or present in the test program, because those dependencies can be "injected" by the interface as mocks or stubs, allowing the behavior of the class under testing to be isolated. The Suicide Hotline application uses the Spring.NET Framework, which is an adaptation of the Spring Framework for Java.

The Data Access Object (DAO) classes, as well as the Spring.NET Web Service proxy class, are "injected" into their dependent classes, for the most part the web page code-behinds, via the Spring.NET Framework. This framework is initialized and launched via the web.config file at root of the web application. The Spring configuration files are located in /Config and /Config/Prod.

Spring.NET also allows for better implementation of the Model/View/Controller concept, by allowing domain classes to be "bound" to the web form that is intended to populate their values. To implement this, the SH aspx pages inherit from Spring.Web.UI.Page instead of System.Web.UI.Page. This allows for implementing and overriding methods in the Spring.Web.UI.Page class that bind a domain object's fields to fields in the form. It also allows for binding data from the database directly to web controls, pre-populating them with data. Binding can be one-way or two-way, depending on what the developer wants. With one-way binding, the form fields can be populated by an object, but any changes to the form will not be reflected in the object, but two-way binding allows changes to go both from object to form and form to object.

For more information on, and examples of the Spring.NET Framework, see http://www.springframework.net/ .

Web Services

Data retrieval and updating to VistA is done via Medical Domain Web Services (MDWS). This is, concisely, VistA data sent and received via XML over HTTP (the **S**imple **O**bject **A**ccess **P**rotocol), from Medical Data Objects (MDO), which is the technology behind VistaWeb. For more information on MDWS, see http://trac.medora.va.gov/wiki/Projects/MDWS

NHibernate

Communication with the SQL Server database, which houses the Suicide Hotline data is done via the third-party, open-source Object-Relational Mapper **NHibernate**. Domain objects are created by this framework which, when populated via simple, object-oriented language ("select house from neighborhood where house.address = '12234 Elm'" for example), can be immediately used in the program, rather than receiving rows and columns of data which then have to be manually "stuffed" into objects each time data is retrieved. When the object changes, it can be save with a simple "flush" command or a “saveupdate.” This reduces programming overhead considerably. NHibernate also handles transactions, in tandem with the Spring Framework, providing easy rollback in event of errors.

Installation Guide

Installation of the Suicide Hotline application modules is very simple. The web server that will host the application needs to have the following software:

- Internet Information Server 6.0
- .NET Framework 3.5 SP1
- ASP.NET 2.0 AJAX Extensions

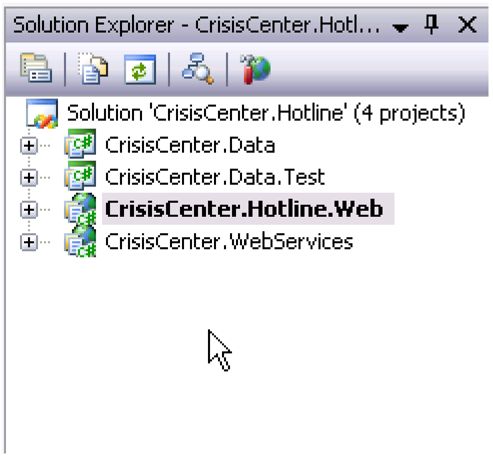
There are five parts to the Suicide Hotline application. They are broken into various “namespaces” and Visual Studio Projects within two solutions to avoid duplication and share code amongst the two modules (Hotline and Response) and to allow for easier testing of the project parts in isolation.

The five parts are:

Project Name and Namespace	**SVN Respository Location**	**Description**
CrisisCenter.Data	http://svn.medora.va.gov/CrisisCenter.Data/trunk	Contains all the Data Access and Domain classes for both Hotline and Response web modules.
CrisisCenter.Manager	http://svn.medora.va.gov/CrisisCenter.Managers/trunk	Contains service classes that facilitate transactional operations in the Response web module.
CrisisCenter.Hotline.Web	http://svn.medora.va.gov/CrisisCenter.Hotline.Web/trunk	The Hotline Web module. This is dependent on the Data and Web Services module and must be loaded into the solution last, if building the project solution for the first time. This is also the project that will be “published” in order to release a build of the web application.
CrisisCenter.Response.Web	http://svn.medora.va.gov/CrisisCenter.Response.Web/trunk	The Response module web solution. This is dependent on the Data and Web Services module and must be loaded into the solution last, if building the project solution for the first time. This is also the project that will be “published” in order to release a build of the web application.
CrisisCenter.WebServices	http://svn.medora.va.gov/CrisisCenter.WebServices/trunk	This is the component that communicates with the MDWS web service to retrieve and write VistA data.

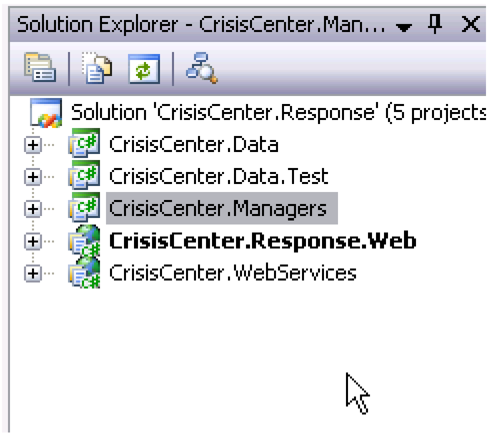
The application should be downloaded from the Subversion source control repository at the paths shown above and loaded into Microsoft Studio.NET 2008 as shown below, to create a solution for both Hotline and Response modules:

Hotline module



Solution 'CrisisCenter.Hotline' (4 projects)
- CrisisCenter.Data
- CrisisCenter.Data.Test
- **CrisisCenter.Hotline.Web**
- CrisisCenter.WebServices

Response module



Solution 'CrisisCenter.Response' (5 projects
- CrisisCenter.Data
- CrisisCenter.Data.Test
- CrisisCenter.Managers
- **CrisisCenter.Response.Web**
- CrisisCenter.WebServices

The projects should be loaded in the order seen above, with the exception that the ".Web" projects should always be brought into the solution last, because they are dependent on the other parts.

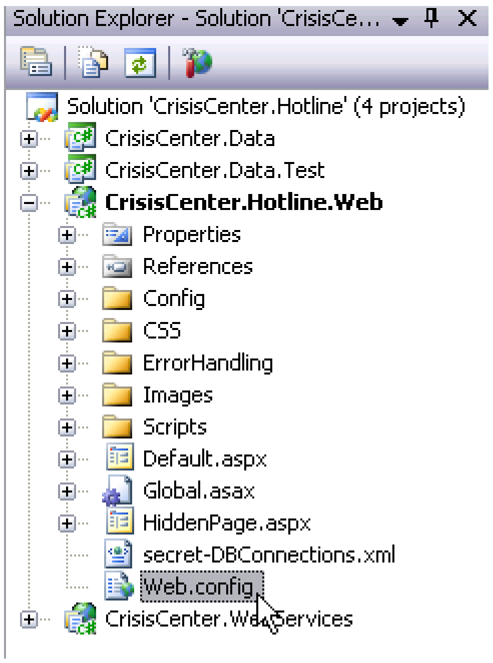
The following document shows how the application should be "published" from the Studio.NET Individual Developer Environment (IDE) using the following guide - http://medora.sharepoint.med.va.gov/Document%20Library/Medora%20Group%20Best%20Practices/Web%20Publishing%20Best%20Practices.doc (Medora Web Publishing Best Practices – located on the Medora Sharepoint Server in Ann Arbor).

The SQL Server Database portion of the application can be created from scripts that also reside in the Medora Group Subversion source control repository at http://svn.medora.va.gov/CrisisCenterDB/trunk

Example of the Spring.NET Dependency Injection And Spring.NET form binding

Here are some screenshots that demonstrate some of the Spring functionality being used within Suicide Hotline.

In the Studio.NET IDE, after loading the project, we will look at the main application configuration file, **web.config**:



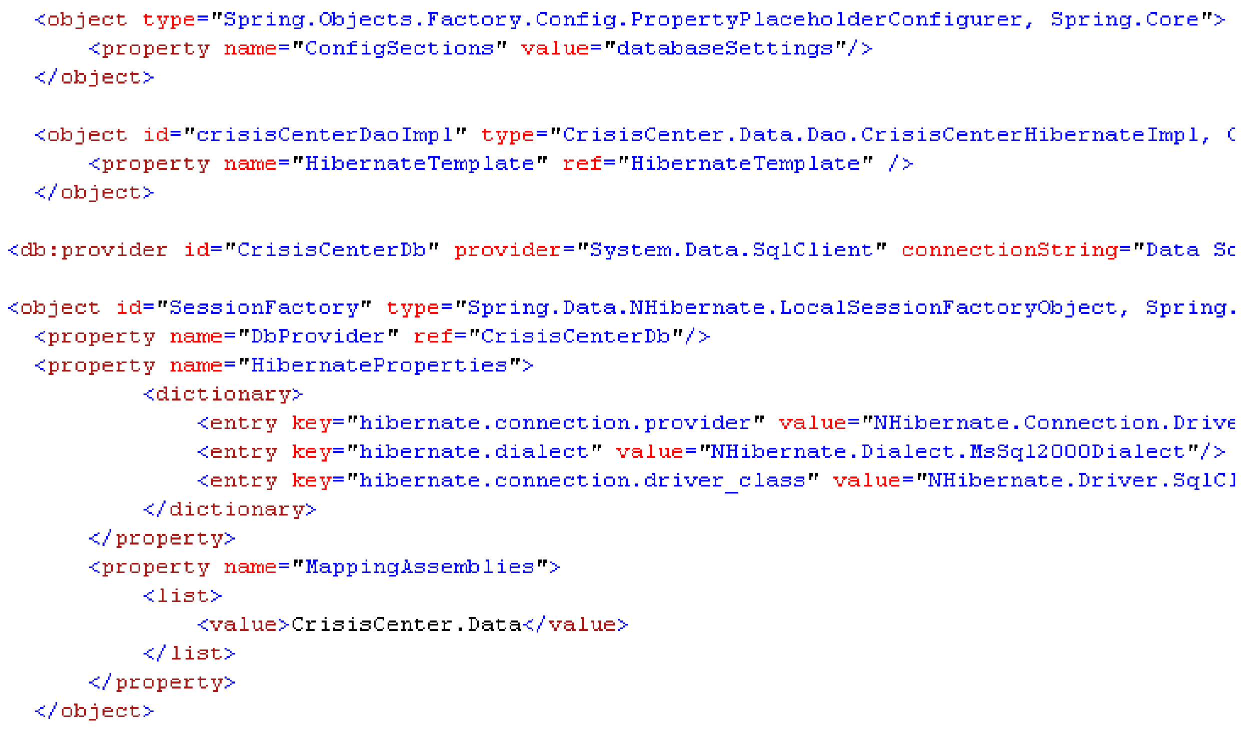
Within the web.config, one can see the settings for the Spring Framework Dependency Injection (DI) feature. Various dependencies of the web page class, such as the Data Access Objects (DAO), Web Service Proxy, and a Service Manager class (Dao.xml, WebServicesDao.xml, SpringWebDI.xml, Services.xml)

```
<spring>
    <parsers>
        <parser type="Spring.Data.Config.DatabaseNamespaceParser, Spring.Data"/>
  <parser type="Spring.Transaction.Config.TxNamespaceParser, Spring.Data" />
</parsers>
    <context>
        <!--
        <resource uri="assembly://crisiscenter/gov.va.medora/WebServicesDao.xml"
        -->
        <resource uri="~/Config/WebServicesDao.xml"/>
        <resource uri="~/Config/Prod/Dao.xml"/>
        <resource uri="~/Config/SpringWebDI.xml"/>
    </context>
</spring>
<databaseSettings configSource="secret-DBConnections.xml"/>
```

The last setting is referencing a configuration file that sets the database server name, database name, and application database user id. This particular file is NOT kept in SVN, since the content of SVN is subject to Freedom of Information Act requests, which would expose security data (user id and password) to the public.

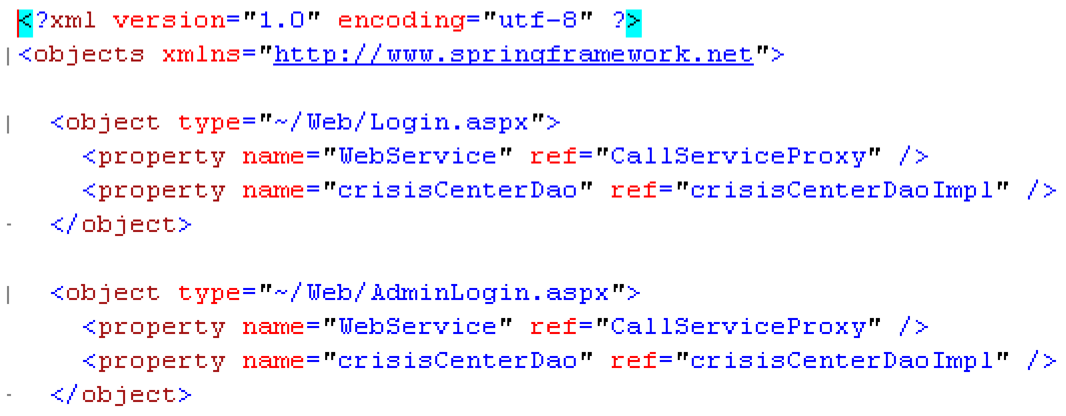
Looking at one of the configuration files below, we see an XML reference to an class within the application, which is used by the Spring.NET Framework to actually generate the object in memory as the application, and hence the Framework, is launched. This object can have certain properties within it set by this file and can also be “injected” into any other object within the Framework. The Framework is made aware of an object by referencing it in one of these Spring.NET configuration files. In this application, Setter methods are used to “inject” on object or value into an object within the Framework.

Dao.xml



```
  <object type="Spring.Objects.Factory.Config.PropertyPlaceholderConfigurer, Spring.Core">
      <property name="ConfigSections" value="databaseSettings"/>
  </object>

  <object id="crisisCenterDaoImpl" type="CrisisCenter.Data.Dao.CrisisCenterHibernateImpl, 
      <property name="HibernateTemplate" ref="HibernateTemplate" />
  </object>

<db:provider id="CrisisCenterDb" provider="System.Data.SqlClient" connectionString="Data S

<object id="SessionFactory" type="Spring.Data.NHibernate.LocalSessionFactoryObject, Spring.
  <property name="DbProvider" ref="CrisisCenterDb"/>
  <property name="HibernateProperties">
          <dictionary>
              <entry key="hibernate.connection.provider" value="NHibernate.Connection.Drive
              <entry key="hibernate.dialect" value="NHibernate.Dialect.MsSql2000Dialect"/>
              <entry key="hibernate.connection.driver_class" value="NHibernate.Driver.SqlC
          </dictionary>
      </property>
      <property name="MappingAssemblies">
          <list>
              <value>CrisisCenter.Data</value>
          </list>
      </property>
  </object>
```

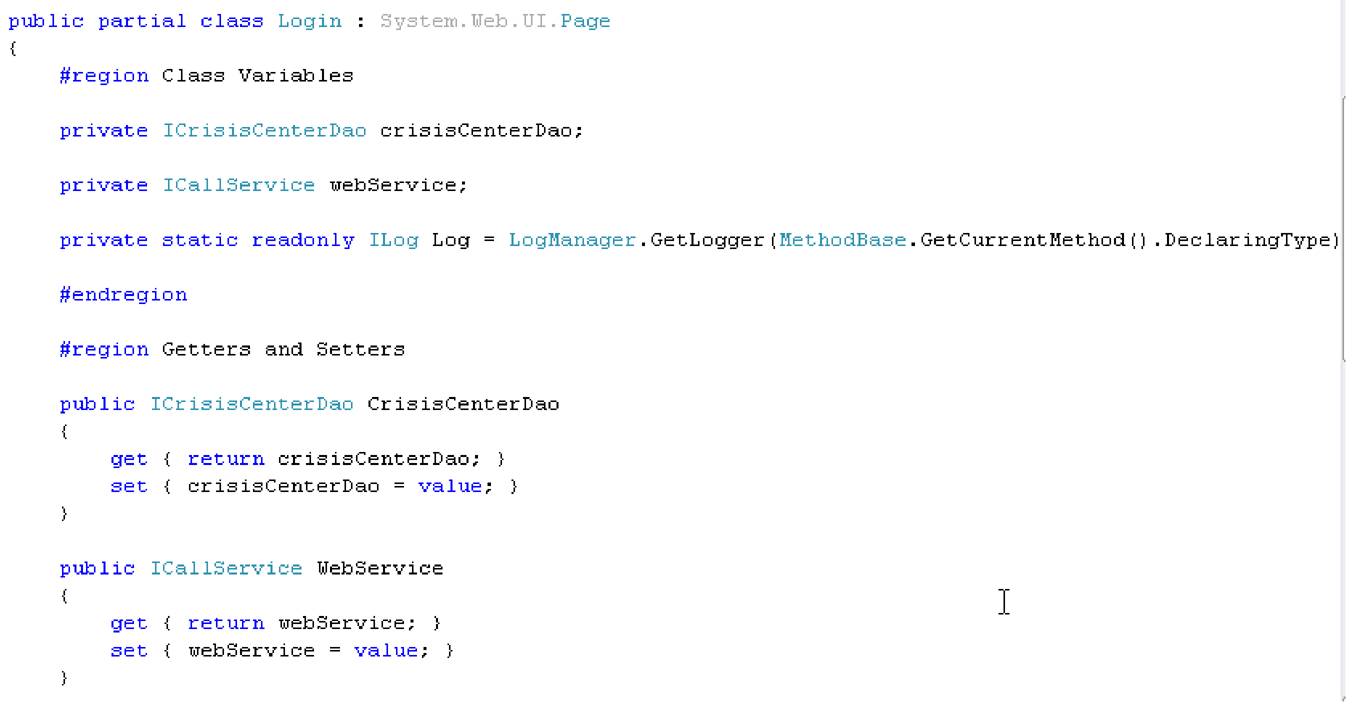
SpringWebDI.xml



```
<?xml version="1.0" encoding="utf-8" ?>
<objects xmlns="http://www.springframework.net">

  <object type="~/Web/Login.aspx">
    <property name="WebService" ref="CallServiceProxy" />
    <property name="crisisCenterDao" ref="crisisCenterDaoImpl" />
  </object>

  <object type="~/Web/AdminLogin.aspx">
    <property name="WebService" ref="CallServiceProxy" />
    <property name="crisisCenterDao" ref="crisisCenterDaoImpl" />
  </object>
```

Dependency Injection at Work in Login.aspx

A look inside the Login.cs class, which is the “Code-Behind” class for the Login.aspx page, we see Spring.NET’s Dependency Injection in action:

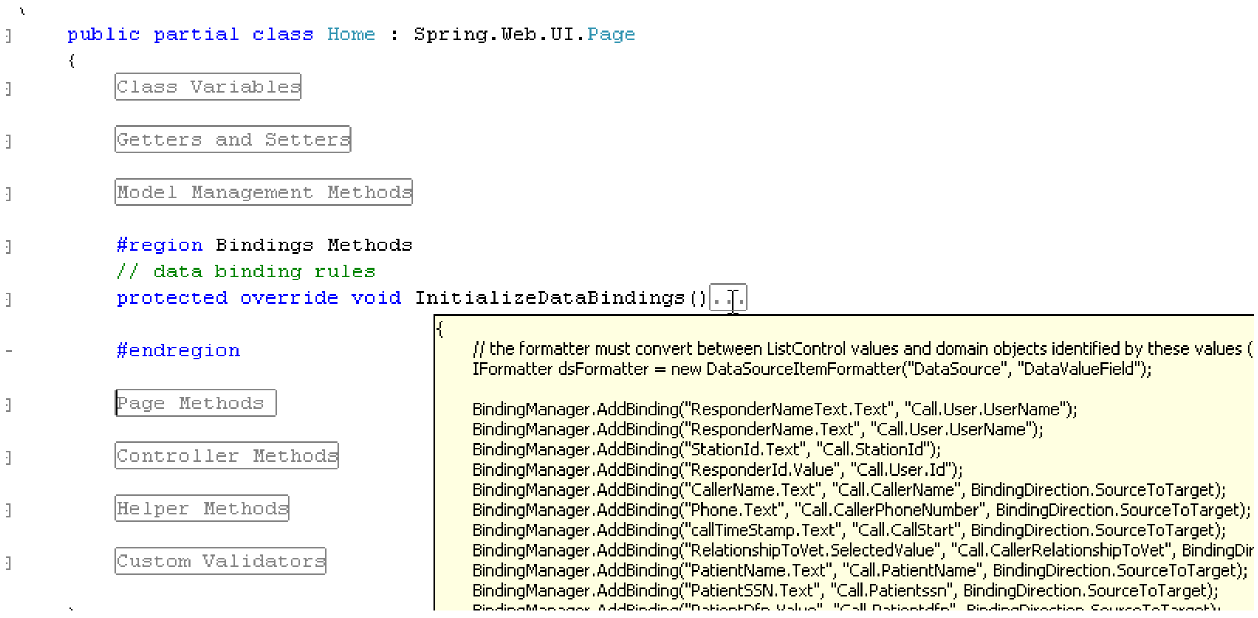


```
public partial class Login : System.Web.UI.Page
{
    #region Class Variables

    private ICrisisCenterDao crisisCenterDao;

    private ICallService webService;

    private static readonly ILog Log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType)

    #endregion

    #region Getters and Setters

    public ICrisisCenterDao CrisisCenterDao
    {
        get { return crisisCenterDao; }
        set { crisisCenterDao = value; }
    }

    public ICallService WebService
    {
        get { return webService; }
        set { webService = value; }
    }
```

The value of the class variable “crisisCenterDao,” which is of the type of an interface to the Dao class “crisisCenterDaoImpl” (shorthand for “Crisis Center Dao Implementation”) is “injected” by the Spring.NET Framework, via the SpringWebDI.xml file with an instance of the class “crisisCenterDaoImpl,” which we see in the Dao.xml config file further above. By doing this, we enable the Login.cs class to call methods of the crisisCenterDaoImpl class without having to instantiate the class within its code via a contstructor, as one would usually do (ICrisisCenterDao dao = new crisisCenterImpl();). This makes coding much simpler, and testing easier. If needed, I can test my Login.cs code without even having a completed Dao class. I can inject a mock class to simulate some results, and as long as my mock implements the interface, the Login.cs page doesn’t “know” the difference, and the code around the Dao references can be tested for validity. Later, when the Dao has been created, it can be injected into the page in place of the mock. Also, if the dao class name or constructor changes for any reason, my Login class will not be affected, as it does not reference the class directly, but only through an interface.

Data Binding in Spring.NET forms

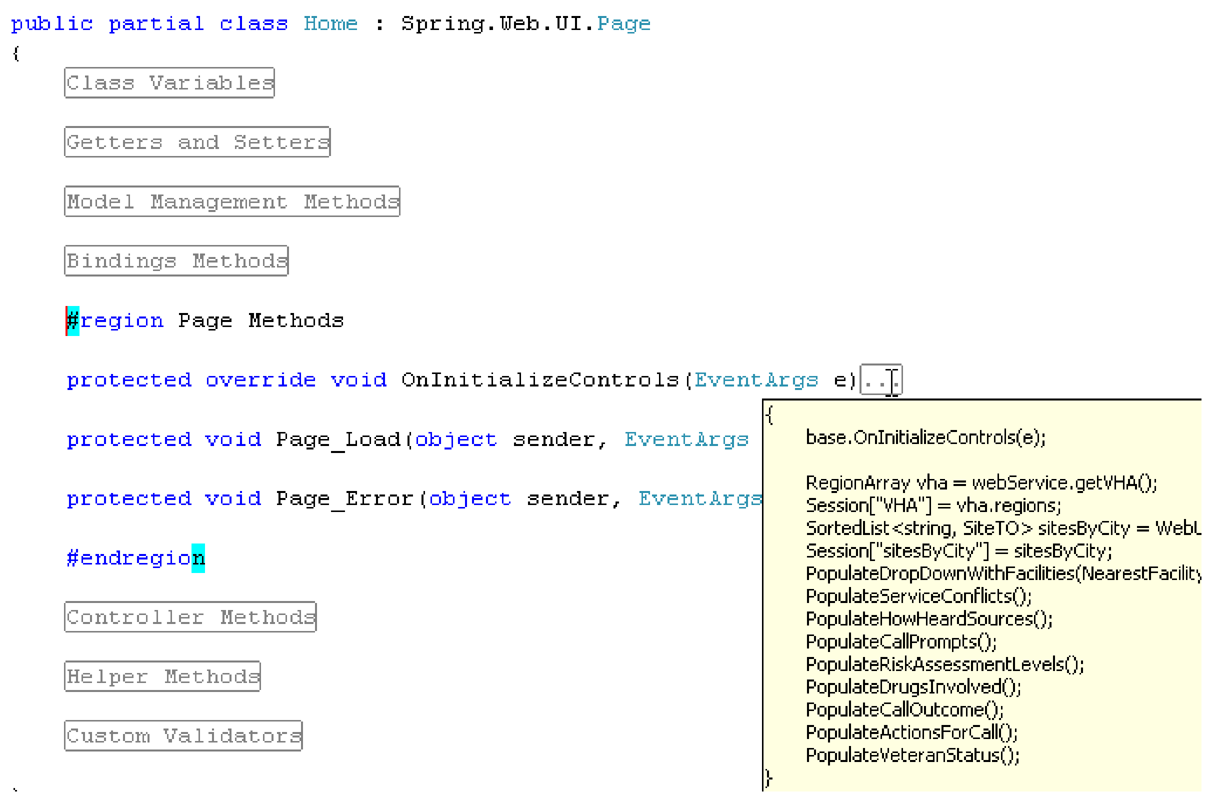
With Spring.NET, you can “bind” a domain object directly to your form, and hence populate it with data from the user’s form entries, or from data drawn from a database and put into the domain object. In this way, reading the result of each line of a form, and manually putting that data into an object becomes unnecessary, shortening development time and streamlining code. Here is an example of the settings needed to Bind a domain class to a form.



```
public partial class Home : Spring.Web.UI.Page
{
    Class Variables

    Getters and Setters

    Model Management Methods

    #region Bindings Methods
    // data binding rules
    protected override void InitializeDataBindings()...

    #endregion

    Page Methods

    Controller Methods

    Helper Methods

    Custom Validators
```

```
{
    // the formatter must convert between ListControl values and domain objects identified by these values (
    IFormatter dsFormatter = new DataSourceItemFormatter("DataSource", "DataValueField");

    BindingManager.AddBinding("ResponderNameText.Text", "Call.User.UserName");
    BindingManager.AddBinding("ResponderName.Text", "Call.User.UserName");
    BindingManager.AddBinding("StationId.Text", "Call.StationId");
    BindingManager.AddBinding("ResponderId.Value", "Call.User.Id");
    BindingManager.AddBinding("CallerName.Text", "Call.CallerName", BindingDirection.SourceToTarget);
    BindingManager.AddBinding("Phone.Text", "Call.CallerPhoneNumber", BindingDirection.SourceToTarget);
    BindingManager.AddBinding("callTimeStamp.Text", "Call.CallStart", BindingDirection.SourceToTarget);
    BindingManager.AddBinding("RelationshipToVet.SelectedValue", "Call.CallerRelationshipToVet", BindingDir
    BindingManager.AddBinding("PatientName.Text", "Call.PatientName", BindingDirection.SourceToTarget);
    BindingManager.AddBinding("PatientSSN.Text", "Call.Patientssn", BindingDirection.SourceToTarget);
    BindingManager.AddBinding("PatientDfn.Value", "Call.Patientdfn", BindingDirection.SourceToTarget);
```

Instead of using the System.Web.UI.Page class Microsoft provides, one uses Spring.Web.UI.Page, which provides several methods to override that allow the data binding and initializing to be done.
The yellow window shows a preview of the code within the binding method. The next graphic shows the method that initializes any controls on the form that are be pre-populated with data from a database.

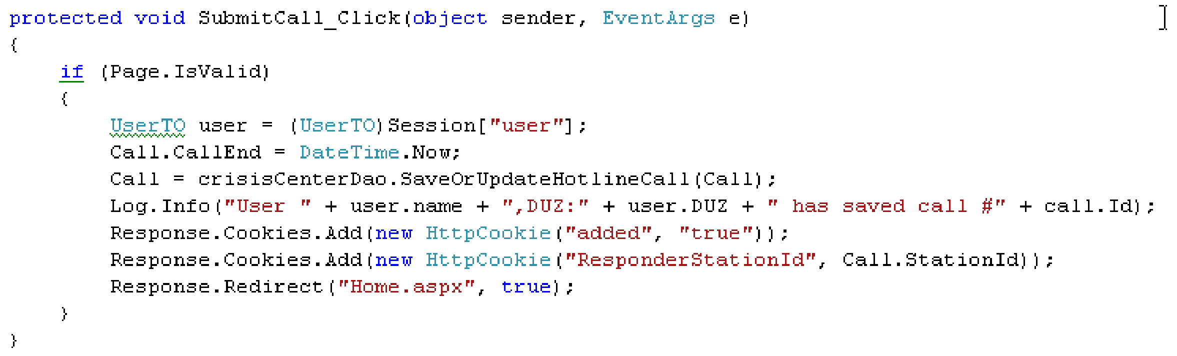


```
public partial class Home : Spring.Web.UI.Page
{
    Class Variables

    Getters and Setters

    Model Management Methods

    Bindings Methods

    #region Page Methods

    protected override void OnInitializeControls(EventArgs e)...

    protected void Page_Load(object sender, EventArgs

    protected void Page_Error(object sender, EventArgs

    #endregion

    Controller Methods

    Helper Methods

    Custom Validators
```

```
{
    base.OnInitializeControls(e);

    RegionArray vha = webService.getVHA();
    Session["VHA"] = vha.regions;
    SortedList<string, SiteTO> sitesByCity = WebL
    Session["sitesByCity"] = sitesByCity;
    PopulateDropDownWithFacilities(NearestFacility
    PopulateServiceConflicts();
    PopulateHowHeardSources();
    PopulateCallPrompts();
    PopulateRiskAssessmentLevels();
    PopulateDrugsInvolved();
    PopulateCallOutcome();
    PopulateActionsForCall();
    PopulateVeteranStatus();
}
```

When the form has been filled out and a submit button clicked by the user, and the application needs to save the data in the form, the operation becomes trivial. The domain object which is bound to this form is

“Call,” which, as can be seen, simply gets fed to the Dao method (saveOrUpdateHotlineCall()) that saves the domain class (via NHibernate, an Object/Relational Mapper) to the database.



```
protected void SubmitCall_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        UserTO user = (UserTO)Session["user"];
        Call.CallEnd = DateTime.Now;
        Call = crisisCenterDao.SaveOrUpdateHotlineCall(Call);
        Log.Info("User " + user.name + ",DUZ:" + user.DUZ + " has saved call #" + call.Id);
        Response.Cookies.Add(new HttpCookie("added", "true"));
        Response.Cookies.Add(new HttpCookie("ResponderStationId", Call.StationId));
        Response.Redirect("Home.aspx", true);
    }
}
```

Application Property Files

Suicide Hotline uses several property files to control its runtime behavior. The files are located on the file system in the root directory:
Web.config – the main property file containing settings for the application.
secret-DBConnections.xml – Connection information to the SQL Server database.

Other Configuration Files

Config/Prod/ contains the Spring configuration files for the DAOs (Dao.xml), while Config/ contains those for the web pages (SpringWebDI.xml), the Dao Manager class (Services.xml), the AOP Transactional objects (DeclarativeServicesAttributeDriven.xml), and the Spring Web Service Proxy (SpringWebServices.xml).

Development Platform

The following infrastructure was used to develop Suicide Hotline on Windows workstations:

Application	***Description***
Studio.NET 2008	C# Web Development IDE and compiler
Internet Information Server 7 with .NET Framework 3.5 SP1 and ASP.NET 2.0 AJAX Extensions	Studio.NET 2008 launches its own virtual server upon running the debugger. No server is needed on developer workstation
Internet Explorer 6	Suicide Hotline is a web application; therefore, developers are required to have VA supported versions of browsers installed.
Source Control Management	Subversion source control system. Hosted at http://svn.medora.va.gov
SQL Server 2005	SQL Server 2005 (or 2000) is needed to host the Suicide Hotline database. Scripts are available from the Medora SVN respository.

Development Projects

SH is comprised of two projects and two modules in our source control management system (Subversion - SVN). The two modules, with their corresponding projects are:

Hotline module:

Crisis-center – the web application for the hotline call center.
Crisis-center-test – the Unit testing classes for the web application.

Response module:

Crisis-center-response – The web application for SPCs and Health Techs.
Crisis-center-response-test – the unit testing classes for the response application.

Development Tools

General development and deployment of SH is done using the Studio.NET 2008 SP3 IDE. No other development platform is necessary, although the open-source Object-Mapping-Class-Generating tool “MyGeneration” was used to generate the NHibernate Mapping files and Domain classes. These can be modified and new ones created by hand or with any number of tools, however.

Business Rule Implementation

Web Security

The SH menu options are created dynamically, based on the roles that a user has assigned to them. Users are not presented with menu links that are outside of their roles.

Data Validation – View Layer

Data entry validations are generally performed by JavaScript in each data entry page. This is done to provide immediate feedback to the user for correction and place focus on the data element responsible for the problem. This also removes the overhead of a complete http post operation just to determine that a data entry field may be invalid. Business rules are not implemented in the pages except where validation of one field depends on the value entered in another field. In the case of Javascript failure, additional validation is done on the server side as well.

Data Validation – Control Layer

Suicide Hotline uses the open source Spring.NET framework to control the flow of the program from one class to another, and one web page to another. XML configuration files located in the Config/* folder control the Framework configuration and population of the application with run-time loaded, framework-generated classes.

Data Validation – Business Layer

The Business Layer in SH consists of ASPX Code-Behind classes, Manager classes, and Hibernate domain classes. The Code-Behind classes validate the business function that a user is attempting to conduct, performs audit trail recordkeeping, and call the appropriate Manager class or Data Access Object (DAO) to conduct the transactions which perform the data creation and updates. The Manager classes call the appropriate data access object (DAO) to interact with the database or VistA.

Data Validation – Model Layer

Model objects in Suicide Hotline are contained in the CrisisCenter.Data.Domain namespace. These objects help control data validation by using private data attributes, multiple constructors, and getter/setter methods.

Final data integrity checks are performed by the NHibernate mappings and database DDL. Most SH DAO classes access only the SQL Server database. These ‘internal’ data DAO’s manage record keys, data format conversions, and conversion of improper null values to valid empty values. The SQL Server database schema DDL contains a significant number of constraints that describe primary key columns, protect referential integrity, and ensure existence of all required (“not NULL”) fields.

Several DAO objects are capable of interfacing with VistA systems over the VistALink connectors. Data which is retrieved from or saved to VistA is handled and validated by these “external” data DAO’s.

Transactions

SH uses Spring AOP Transactions to wrap all create and update functions. This allows for simple rollback of all steps in a failed transaction or full commit for successful transactions. SH does not use transactions for query only functions. SH does not have any record deletion capability accessible from the user interface.

Transactions can only performed by the running Suicide Hotline application (system user) or a fully logged in and authorized Suicide Hotline user account. All transactions are logged to the Log4Net text log with the identity of the user who performed it.

Transactions and the associated audit trail record cannot be undone or deleted through any UI provided facility.

Concurrency

Suicide Hotline does not currently account for concurrency, as the same record is not likely to ever be modified by more than one person at a time. The records are modified in a workflow process, with one person handling a record after it has been “passed” to them from another person earlier in the workflow. Once the record passes out of one person’s workflow, they no longer have access to it unless it is “passed back,” as when a Health Technician attaches a note to an incomplete Suicide Prevention Coordinator record, and the SPC needs to change the record.

Glossary/Acronym List

Term/Acronym	*Description*
HT/Health Tech	Health Technician responsible for monitoring follow-up for suicide hotline calls
SPC/Suicide Prevention Coordinator	Clinician at VA facilities who is responsible for follow-up care/counseling of suicidal veterans
ADPAC	Automated Data Processing Application Coordinator
API	Application Program Interface
Audit Trail	A history of the changes made to a record including old data, new data, and the name of the user who made the change. Record of access and modifications
CPRS	Computerized Patient Record System. A VistA software application that provides an integrated patient record system for use by clinicians, managers, quality assurance staff, and researchers
Graphical User Interface (GUI)	A type of display format that enables users to choose commands, initiate programs, and other options by selecting pictorial representations (icons) via a mouse or a keyboard.
HIPAA	Health Insurance Portability and Accountability Act of 1996. Also referred to as, HIPAA.
HTTP	Hyper Text Transfer Protocol
HTTPS	Secured HTTP Protocol
MVC	Model View Controller
OED	Office of Enterprise Development
RDBMS	Relational Database Management System
SSL	Secure Socket Layer
SSO	Single Sign On
TCP/IP	Transmission Control Protocol/Internet Protocol
User	An Administrator, a Clinician, or a Researcher
VA	Veterans Affairs
VAMC	Veterans Affairs Medical Centers
VHA	Veterans Health Administration
VISN	Veterans Integrated Service Network
VistA	Veterans Health Information Systems and Technology Architecture

Index