VistAWeb

Version 16.0
(Patch WEBV*1*25)

Technical Manual

July 2012

Product Development
Office of Information and Technology
Department of Veterans Affairs

Revision History

Date	Patch	Page(s)	Change(s)	Project Manager	Technical Writer
July 2012	WEBV*1*25 VistAWeb 16	Throughout	Updated section on new features in Version 16.	Pam O'Reilly	Nick Metrokotsas
12/11	WEBV*1*23/24	Various	Various edits from Steve Monson	Richard Muse	J. Green
9/11	WEBV*1*23/24	Various	Various changes from Steve Monson	Richard Muse	J. Green
6/11	WEBV*1*23	Various	Various changes from Steve Monson	Richard Muse	J. Green
6/10	WEBV*1*20	Various	Various changes from Steve Monson	Jack Schram	J. Green
5/10	WEBV*1*18	Various	Various changes from Steve Monson	Richard Muse	J. Green
1/6/10	WEBV*1*19	Various	Various changes from Steve Monson	E. Ridley	J. Green
9/1/09	WEBV*1*17	Various	Various changes from Steve Monson and Robert Troha	E. Ridley	J. Green
12/23/08	WEBV*1*16	Appendix D	Added SYSINFO Reports	E. Ridley	J. Green
10/3/08	WEBV*1*13	Various	Revised some URL links, corrected some descriptions, corrected a script, added index creation scripts.	E. Ridley	J. Green
12/04/07	WebV*1*11		Various changes from Steve Monson	S. Madsen	A. Fretz
11/28/2007	WebV*1*11		Removed Setup script for move to Installation Guide.	S. Madsen	A. Fretz
10/24/07	n/a	17-22	VistAWeb scripts.	Scott Madsen (from Steve Monson)	A. Fretz
10/18/07	n/a	16-17	Added section BSE Code Update.	Scott Madsen (from Steve Monson)	A. Fretz

Date	Patch	Page(s)	Change(s)	Project Manager	Technical Writer
February 2007	n/a	p.1	Added note about internal VA network links.	S. Madsen	M. Kelsey
December 2006	WEBV*1*8 VistAWeb 6.0	n/a p. 1 p. 6 and p. 7 p. 7	Various minor text edits. Minor change to Introduction to better reflect functionality and access to VistAWeb. Added mention of access to VistAWeb through the Remote Data Available button in CPRS. Added note about requirement for updating Verify codes by standalone users.	S. Madsen	M. Kelsey
7/11/06	WEBV*1*7	2, 4, 5, 15 App A App B	Removed references to VistAWebDocs application. Added two new scripts to App. A. Removed appendix B, which makes the former appendix C now B.	S. Madsen	M. Kelsey
3/20/06	WEBV*1*5	15, App B	The *Using VistAWeb User Management Web Application* section was modified to reflect a new method of requesting Special User access. Some new file names (CAPRI files, for example) have been added to Appendix B; *testSecurityKey.java* filename was deleted.	S. Madsen	M. Kelsey
5/05/05	n/a	All	Replaced URLs and made format changes.	G. Smith	M. Kelsey
2/25/05	n/a	All	Reviewed for release		
2/18/05	n/a	2, 6, & 28	Removed URL references (VISN CIO will provide)		
2/10/05	n/a	14, 16, 21, & 26	Removed reference to Special User Script		
1/31/05	Informational Patch number OR*3*230	All	Initial User Manual for use with beta test version		

Table of Contents

List of Figures

Introduction

Veterans Health Information Systems and Technology Architecture (VistA) VistAWeb is an intranet web application used to review remote patient information found in VistA, the Bi-Directional Health Information Exchange (BHIE) system, the Health Data Repository II (HDR II) databases, and the Nationwide Health Information Network (NwHIN, see http://healthit.hhs.gov/portal/server.pt?open=512&mode=2&cached=true&objID=1142).

To a large extent, VistAWeb mirrors the reports behavior of the Computerized Patient Record System (CPRS) and Remote Data View (RDV). However, by permitting a more robust and timely retrieval of remote-site patient data, VistAWeb is also an enhancement to CPRS/RDV.

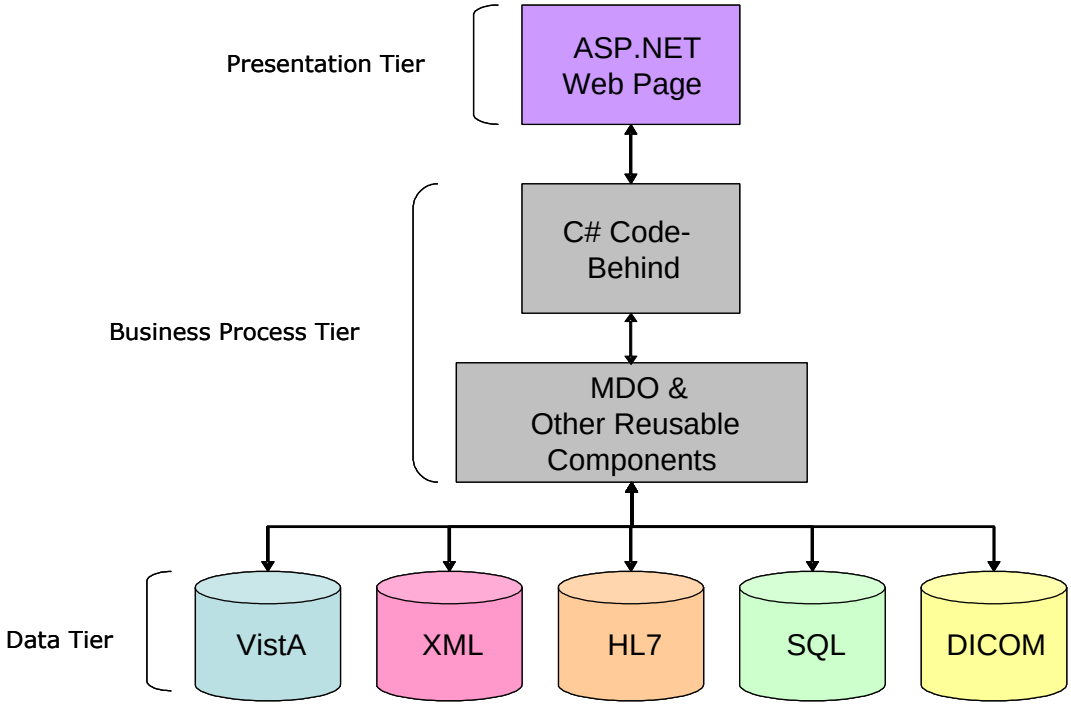
There are three ways to access VistAWeb. VistAWeb can be made available by adding it to the CPRS Tools Menu, and it can be selected as the default method of retrieving data from the VistaWeb button in CPRS. These two methods are referred to as CPRS-spawned versions of VistAWeb. They are compliant with the Health Level 7 (HL7) Clinical Context Object Workgroup (CCOW) standards and therefore maintain context with the patient selected in CPRS. As a third option, VistAWeb can be accessed in a standalone mode by entering the uniform resource locator (URL) link (https://vistaweb.med.va.gov/) in the Internet Explorer address bar.

Note: *Some links found in this technical manual go to sites or pages found on the VA intranet. These sites or pages are not accessible from outside the VA network.*

The standalone version of VistAWeb is connected to neither CPRS nor the clinical context management application. Standalone VistAWeb serves an important function for users who have been granted special access to multiple sites, such as for National Programs, Veterans Administration (VA) researchers, and others. VistAWeb was also made available more broadly, though temporarily, to assist clinical staff with the retrieval of patient information from the sites affected by damage caused by hurricane Katrina.

The design of VistAWeb uses n-tier architectural principles, where VistAWeb represents the presentation tier (ASP.Net Web Page). The business process tier is represented by multiple components that VistAWeb uses to access the data tier. Business process components include such elements as code-behind pages (a programming feature of Microsoft.NET programming), Medical Domain Objects (MDO), and a collection of other reusable components. The code-behind pages (.cs files), which are directly interwoven within the VistAWeb application, will be covered in this document, while other reusable components (.dll files) such as MDO will be discussed in other technical documents. The data tier comprises multiple data sources, such as VistA, XML, and numerous SQL-compliant relational databases (e.g., Oracle, Microsoft SQL Server, and Microsoft Access). Although some code-behind pages do interact with an SQL-compliant database and some XML sources, the rest of the data tier interactions take place in reusable components such as MDO. Figure 1 depicts, at a high level, the interaction of the different application tiers.

Figure 1: VistAWeb Application Tiers Interaction



Assumptions

System administrators, specifically web administrators, are the intended audience of this technical manual. Readers are assumed to possess the technical knowledge of programming principles using languages such as Java, XML, C#, HTML, and SQL. Additionally, users of this manual should also possess the technical knowledge of how to configure and interact with application servers and are also assumed to have already implemented the necessary security hardening guidelines. See the Office of Cyber and Information Security link below for information pertaining to security requirements:
https://vaww.infoprotection.va.gov/nsoc/default.aspx

For additional information on technologies and principles used in the development and implementation of VistAWeb, the following sources are recommended reading:

ASP.NET: http://www.asp.net
C#: http://msdn.microsoft.com/en-us/vstudio/hh388566
C# Design Patterns: http://msdn.microsoft.com/en-us/magazine/cc301852.aspx
.NET Application Development / n-tier: http://msdn.microsoft.com/en-us/library/ms973279.aspx
SQL: http://www.w3schools.com/sql/default.asp
World Wide Web Consortium: http://www.w3.org/
Spring Framework: http://www.springframework.net/

Log4net (similar to Log4j): http://logging.apache.org/log4net/index.html (see also Chainsaw at http://logging.apache.org/chainsaw/index.html)

The remainder of this document is divided into the following sections:

- System Requirements
- VistAWeb Overview
- SQL Server Database Overview
- Appendixes

System Requirements

Hardware

The servers that run VistAWeb are configured in Austin, TX. The exact details of the web and SQL servers can found in the following links:

Web Server:
http://vaww.sms.aac.va.gov/SMSReporting/MachDetails.asp?Machine=vaausnvwweb200

SQL Server:
http://vaww.sms.aac.va.gov/SMSReporting/MachDetails.asp?Machine=VAAUSNVWSQL200

Software

VistAWeb is capable of running on either Windows Server 2003 or Windows Server 2008. Each configuration is listed below.

Windows Server 2003

Windows Server 2003 Enterprise configured with the role of Application Server
Internet Information Services (IIS) 6.0 (installed by default as part of the Application Server role)
.NET Framework 2.0
FTP services and an FTP folder (to be used as a staging location for updates to VistAWeb)
Web Extension Services set to allow ASP.NET extensions (see Figure 2)
Registry entries in Windows 2003 Server that allow more ephemeral ports (see Figure 3)
SQL Server 2000 or higher (The database does not need to run on the same server as the web application.)

Windows Server 2008

Windows Server 2008 R2 Enterprise configured with the role of Web Server
Internet Information Services (IIS) 7.5 (installed by default as part of the Web Server role)
.NET Framework 2.0 (installed by default in Windows 2008)
SQL Server 2005 or higher (the database does not need to run on the same server as the web application.)

Figure 2: Web Service Extension Settings in Windows Server 2003

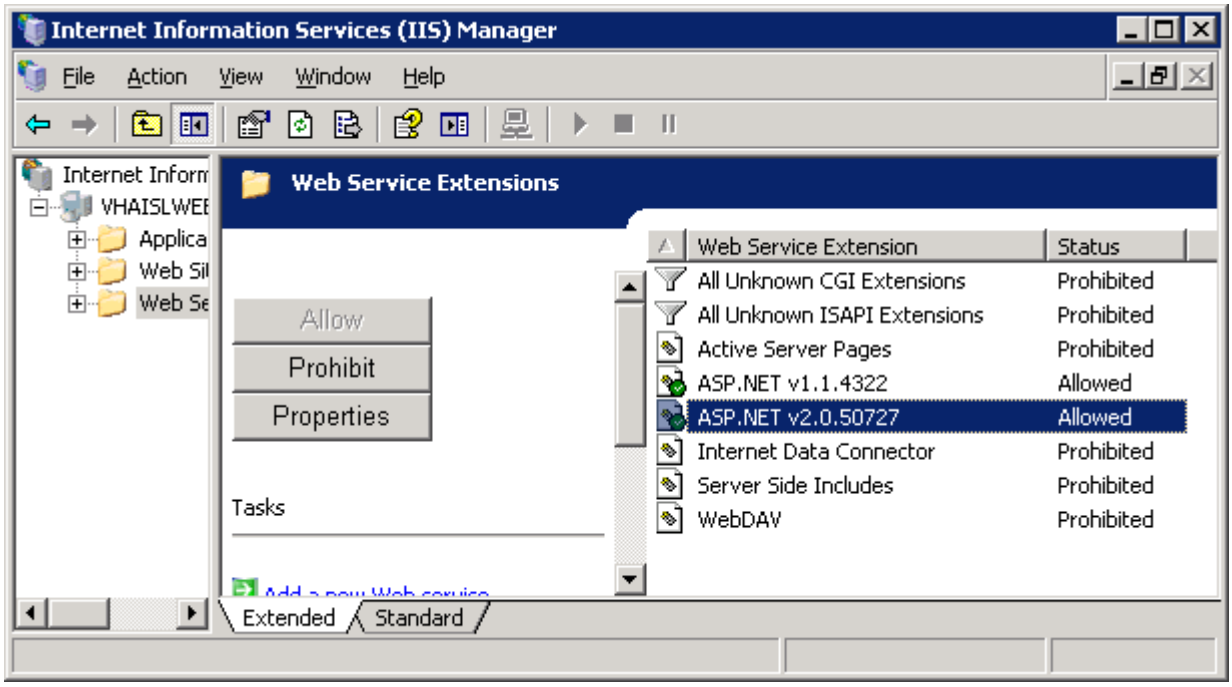
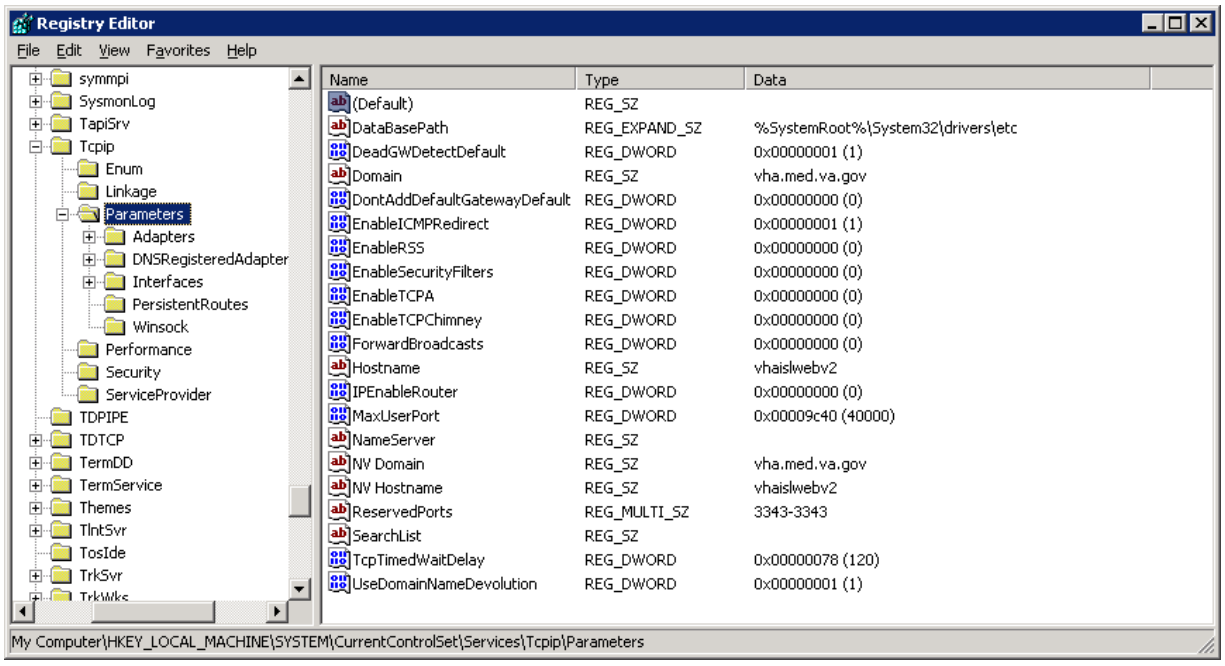


Figure 3: MaxUserPort and TcpTimedWaitDelay Registry Settings in Windows 2003 Server



For instructions on installing and configuring VistAWeb and the related intranet application, see the VistAWeb Installation Guide.

VistAWeb Overview

VistAWeb's web pages are a mix of basic HTML pages and .NET Active Server Pages (ASP.NET), noted by the respective extensions of .htm (or .html) and .aspx. HTM and HTML pages are standalone pages. However, for the ASP.NET pages, there are actually two pages to be aware of:

aspx pages are written in HTML and ASP.NET tags;
aspx.cs pages represent the "code-behind" pages that power the aspx pages.

Note: *A code sample is provided in Appendix B, which contains a sample code from each of the three—htm, aspx, aspx.cs.*

Language Specifications

The languages used in VistAWeb (and also VistAWeb Context) are:

ASP.NET
HTML, including Cascading Style Sheets (CSS)
XML
C#
JavaScript

Using VistAWeb

VistAWeb is an intranet application that may be accessed as a standalone application, or spawned from the CPRS tools menu or the CPRS VistAWeb button. VistAWeb's patient data menu is very similar to the report menu found in the CPRS Reports tab. Users may wish to use either the standalone application or the CPRS-spawned version from the CPRS Tools menu. VistAWeb users must have an active existing CPRS account with the necessary CPRS contexts enabled. The two methods for using VistAWeb are documented below. For more comprehensive documentation on the use of VistAWeb, please read the VistAWeb User Manual.

Standalone Application Process

1. User launches the web browser application (e.g., Internet Explorer).
2. User enters the URL of VistAWeb (https://vistaweb.med.va.gov) in the address bar and presses the Enter key or clicks the mouse cursor on the "Go" button adjacent to the address bar.
3. VistAWeb loads into the user's web browser.
4. User must select a login site link on the left of the display screen by clicking the mouse cursor on the desired site where the user has access.
5. User is provided with the VistA Login screen.
6. User enters his or her CPRS access/verify codes in the spaces provided and presses the Enter key or clicks the mouse cursor on the Login button.

7. Once the user's account is authenticated against CPRS, the user's remote site patient selection permissions are verified from a SQL Server database. The permissible sites for the user to choose patients from are then displayed.
8. User must select a site from which to select a patient if selecting a site other than his or her default site.
9. User is presented a Patient Selection screen for entering the desired patient name (last name, comma, first name, and middle initial), portion of name, first initial of last name and last 4 digits of Social Security number, similar to patient selection in CPRS. User then clicks the mouse cursor on the "Find" button (or presses the Enter key) to find a list of patients matching the criteria. With the appropriate patient highlighted, user then clicks on OK (or presses the Enter key).
10. The Sites and Notices screen is displayed, which identifies the sites where the patient has been seen.
11. User can look at different elements of the patient record by selecting a desired report from the list of available reports on the left side of the displayed screen.

By default, a VistAWeb user is permitted to select patients that are in the local VistA system where the user logs in. VistAWeb will retrieve data for these patients from all sites where the patients have visited. Some users (researchers or referral coordinators, for example) may need to select patients that are not in the local VistA. These users must be granted Special User access. Special User access can be granted for one site in addition to the login site, several sites, an entire VISN, or all sites nationally. The process for requesting special user permission is documented in the VistAWeb User Manual.

Note that regardless of which site is selected for a patient (local or remote), once a patient has been selected, VistAWeb uses the Master Patient Index (MPI) at the selected site to determine at what other sites the patient has remote data and establishes the connections to each of those sites to retrieve data requested by the user.

Note: *Users who regularly only use the standalone version of VistAWeb will be required to update their verify codes periodically, just as they would if logging into CPRS. When this happens, the login screen will display the message, "User must enter a new Verify code at this time."*

CPRS-Spawned VistAWeb Process

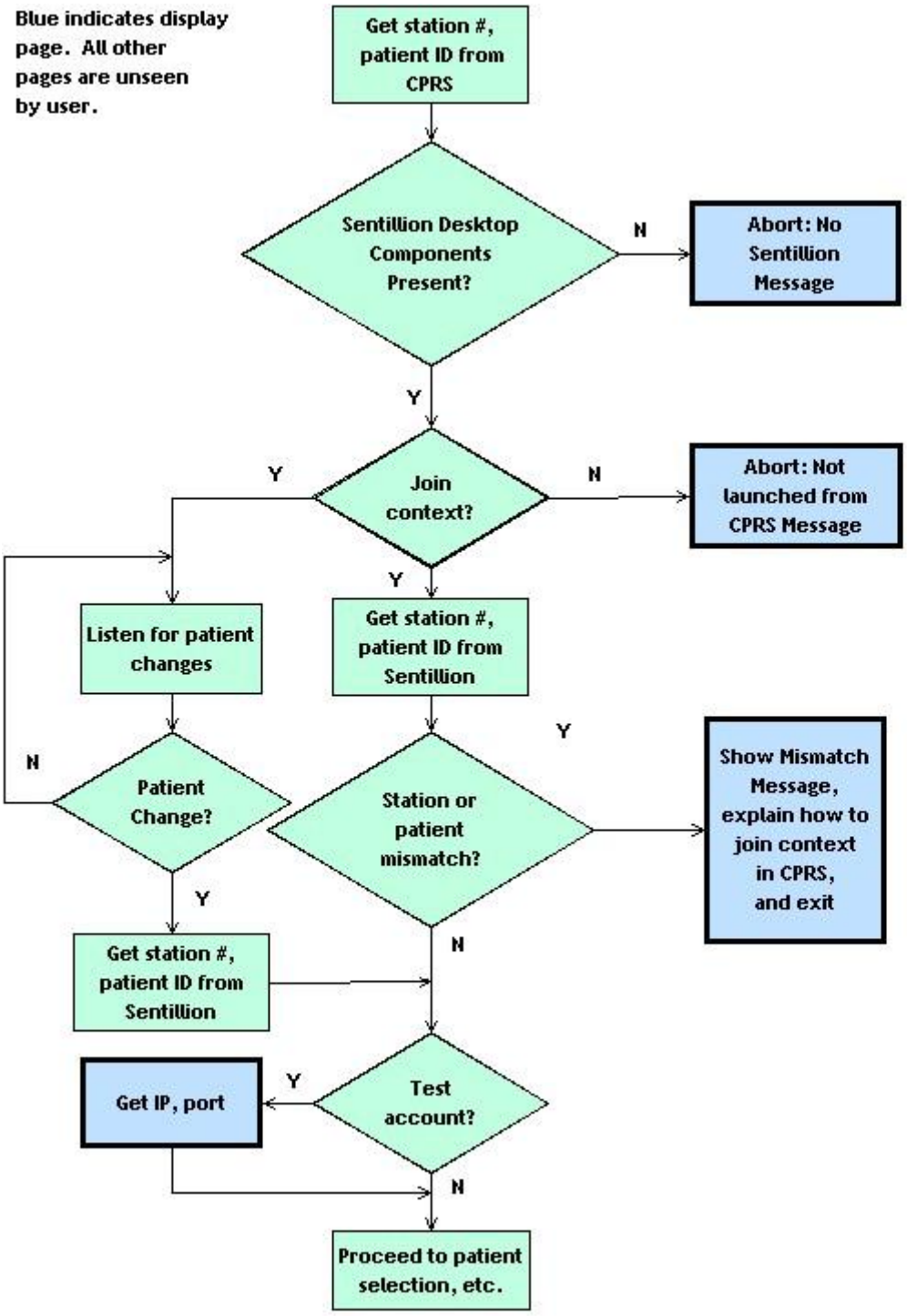
1. User opens CPRS.
2. User enters his or her CPRS access/verify code.
3. User selects a patient.
4. User launches VistAWeb from the Tools menu or the CPRS VistAWeb button
5. VistAWeb opens in a web browser window (e.g., Internet Explorer) and one of two things occurs:
   - If the proper Sentillion Vergence controls are installed, VistAWeb will synchronize and display the CPRS-selected patient if the session of CPRS that launched VistAWeb is in context. The user can now look at different elements of the patient record by selecting the desired report from the reports menu on the left side of the user's display screen.

- If the components are not installed, or if the CPRS session is not in context, then the user is warned that either the components are not installed, or that CPRS lacks proper context synchronization, and the user is forced to close VistAWeb. After either or both of these situations are rectified, the user can start the process anew and display the patient records using VistAWeb, as in number 1 above.

Note: *Unlike the standalone version of VistAWeb, the CPRS- spawned version does not permit the user to change patients from within VistAWeb. VistAWeb is CCOW compliant and, therefore, maintains context with the patient who was selected in CPRS. Users must have the Sentillion Vergence Locator application loaded on the desktp and must maintain patient synchronization (context) to use the CPRS-spawned VistAWeb process.*

Figure 4 shows the process flow for the login and use scenarios discussed above.

Figure 4: VistAWeb Login Process Flow



VistAWeb Under the Hood

Like all applications developed using Active Server Pages (ASP), all data source communications and most complex tasks are performed on the server side rather than the client side. The client (i.e., the VistAWeb user) only sees the end result of what the server does. What the client sees is standard HTML and client-side scripting generated by the server. So even though a common ASP.NET tag of <asp:textbox> might exist in an aspx page on the server, what is shown in the client's browser is <input type=text>. More importantly, the client never sees or knows how the data is collected or from where it is collected.

VistAWeb communicates with VistA and other data sources using a collection of methods. All data source communication methods are done through code-behind pages and through reusable components. Both the code-behind pages (.cs files) and the reusable components (.dll files) represent the business process tier. To provide better context and clarity, let's examine a code-behind page and a web page in greater detail.

ASP.NET / Code-Behind Example: HsAdhoc.aspx

HsAdhoc.aspx is a basic ASP.NET page, but the coding techniques are the same for most ASP.NET pages in VistAWeb. Note that only the unique elements of ASP.NET will be explained here.

```
<%@ Page language="c#" Inherits="EMR.HsAdhoc" CodeFile="HsAdhoc.aspx.cs"
%>
```

Every .aspx page must identify which code-behind page it must link to. In this case, it is Insurance.aspx.cs. It is possible to use C# programming in-line the same way that VBScript and JavaScript can be used in regular ASP (i.e., pre-.NET) pages, but VistAWeb only uses C# in code-behind pages.

```
<form id="Form2" method="post" runat="server">
                        <!-- Adhoc Table
************************************************************-->
                        <table border="0">
                                <tr> <!-- Row 2-->
                                        <td></td>
                                        <td>Select Adhoc Reports:<asp:label id="lblAdhoc"
runat="server"></asp:label></td>
                                        <td></td>
                                        <td>Component Selection(s)<asp:label id="lblCompSel"
runat="server"></asp:label></td>
                                        <td></td>
                                        <td><asp:label id="lblHdrName" runat="server"
Enabled="False">Header Name:</asp:label></td>
                                </tr>
                                <tr> <!-- Row 3-->
                                        <td></td>
                                        <td vAlign="top" rowSpan="6"><asp:listbox id="lstAdhocRpts"
Width="250px" Runat="server" Height="180px"
onselectedindexchanged="lstAdhocRpts_SelectedIndexChanged"></asp:listbox></td>
                                        <td></td>
                                        <td vAlign="top" rowSpan="6"><asp:listbox id="lstSelected"
tabIndex="4" Width="250px" Runat="server" Height="180px" AutoPostBack="True"
onselectedindexchanged="lstSelected_SelectedIndexChanged"></asp:listbox></td>
                                        <td></td>
```

```
                                    <td><asp:textbox id="txtHdr" tabIndex="7" Width="150"
Runat="server" AutoPostBack="True" ontextchanged="txtHdr_TextChanged"></asp:textbox></td>
                                </tr>
                                <tr> <!-- Row 4-->
                                    <td></td>
                                    <td><asp:button id="btnAdd" tabIndex="1" runat="server"
Width="100%" Enabled="False" Text=">" CssClass="myButton"
                                                    CausesValidation="False"
onclick="btnAdd_Click"></asp:button></td>
                                    <td><asp:button id="btnRptUp" tabIndex="5" runat="server"
Width="100%" Enabled="False" Text="Up"
                                                    CssClass="myButton"
onclick="btnRptUp_Click"></asp:button></td>
                                    <td colSpan="1"><asp:label id="lblOccLimit" Runat="server"
Enabled="False">Occurrence Limit:</asp:label></td>
                                    <td><asp:label id="lblTimeLimit" Runat="server"
Enabled="False">Time Limit:</asp:label></td>

                                </tr>
```

The ASP.NET web controls, when read by the server, are rendered as HTML tags when transmitted to the client. For example, the `<asp:listbox>` tag will be rendered as a `<select>` tag. The `<asp:button>` tag will be rendered as an`<input type="button"...>`. Note that ASP.NET web controls can be read by the C# code-behind pages before the page is rendered, while the regular HTML tags cannot. This is because all ASP.NET and C# code is interpreted first, and HTML controls are interpreted last. In fact, HTML controls cannot be read by the C# code, because the controls do not exist prior to the page being loaded. The code-behind is executed prior to the .aspx page.

At the beginning of every code-behind page, there are usually several lines of code that identify which additional class libraries should be included and referenced. Including these libraries permits the programmer to interact with the routines within the libraries. For example, the System.Web.UI.WebControls Library allows the programmer to interact with the methods and properties of the ASP.NET controls and receive programming assistance from the IDE syntax checker.

A major difference between ASP and ASP.NET pages is that ASP.NET pages are self-submitting, meaning that they do not submit their data to other ASP.NET pages. In the pre-.NET days, ASP pages usually submitted their data to a different ASP page, with the different page being identified in the HTML tag <form onaction="secondasppage.asp" method="post">. In ASP.NET, the onaction event of the form tag is ignored, and clicking a submit button will submit the form's data to itself. With this paradigm shift, additional programming elements were needed for the code-behinds to prevent certain pitfalls. For example, say the Insurance.aspx page is being visited for the first time by the user. Certain values are queried and displayed on the page. Without a conditional element being included in the Page_Load event, any reload would continually reload the original values, thereby wiping out any alterations to the page made by the user (assuming any were allowed). This conditional element sample is identified below:

```
private string summaryTitle;
private string summaryID;
private string siteId;
protected ArrayList uComponents = new ArrayList();
protected int compCtr;
```

```
protected void Page_Load(object sender, System.EventArgs e)
{
        base.pageLoad(sender, e);
        summaryTitle = Request.QueryString["alt"];
        summaryID = Request.QueryString["app"];
        siteId = Request.QueryString["siteId"];
        MDO.MultiSiteDAO dao = (MDO.MultiSiteDAO)Session[Constants.MULTI_SITE_DAO];

        if (!Page.IsPostBack)
        {
                Session["AHComponents"] = uComponents;
                MDO.AHComponent[] ahComps = dao.getAdhocComponents(siteId);
                for (int i=0; i<ahComps.Length; i++)
                {
                        ListItem item = new
ListItem(ahComps[i].getComponentName(),ahComps[i].getAHString());
                        lstAdhocRpts.Items.Add(item);
                }

        }
        if (lstAdhocRpts.Items.Count>0)
        {
                btnAdd.Enabled = true;
        }
        uComponents = (ArrayList)Session["AHComponents"];
        compCtr = 0;
        if (uComponents.Count > 0)
        {
                compCtr = int.Parse(((MDO.AHComponent)uComponents[uComponents.Count-
1]).getCompCtr());
        }
}
```

Having the !Page.IsPostBack condition ensures that the setup code is performed only once rather than each time the page is reposted to itself.

Releasing Project Updates to Production

Whenever there is a change to the VistAWeb EMR project or code files (C# files), the project must be recompiled. This project recompilation produces a .dll file. This .dll file, unlike the pre-.NET days of ASP and middle-tier development, does not require a component object model (COM) wrapper, but is instead COM-less. Bottom line—this means faster application development, faster deployment, and less application server downtime. Unlike the pre-.NET days when the IIS services would have to be stopped to release .dll updates, in .NET, the IIS services do not need to be stopped, because the COM-less .dll is not locked and can be easily overwritten.

Despite this feature, VistAWeb is always released in a complete zip file to ensure that all components are version compatible with each other, and that they have all passed SQA testing as a unit.

Other VistAWeb Mannerisms

CCOW—VistAWeb is Clinical Context Object Workgroup (CCOW) compliant and therefore maintains context with the patient who was selected in CPRS. VistAWeb

retrieves data for that patient from all sites where the patient has visited. Users will not be able to select a new patient from within VistAWeb, but may return to CPRS to select a new patient. This new patient will then be viewable in VistAWeb automatically.

Activity Logging—VistAWeb tracks the user's movements through the application in a SQL Server database table.

Error Logging / Email—whenever an error occurs with the application, it is automatically logged by VistAWeb to the log4net framework. The log4net framework can be configured to save log entries to a file, the SQL Server, emailed, etc. VistAWeb comes configured to store all log messages to the SQL Server database.

Multithreaded Site Connections—when a user identifies what patient data he/she wants to see, MDO creates a separate connection thread to each site where the patient has data and retrieves the data asynchronously rather than iteratively.

No Caching—pages that present patient data are prevented from being cached in the client-side Temporary Internet Files folder, thereby preventing users from retrieving patient data after the VistAWeb session has terminated.

Session Timeout—the VistAWeb browser session times out after 15 minutes of inactivity. A two-minute warning window will pop up, allowing the user the option to terminate the session early or continue working, the latter choice thereby resetting the timeout period.

Spring Context—much of VistAWeb is configured using Spring framework configuration files

Special User Access Web Application

The Special User Access application is an intranet application that is used by select individuals to grant VistAWeb users the ability to select patients from remote sites, and is built in to VistAWeb. Note that access to this application is restricted to only a handful of individuals who have been identified to have the proper credentials necessary to assign remote site permissions to VistAWeb users.

Process

Once a VistAWeb user has submitted a request to be able to select patients from one or more remote sites, the associated authorizer will use the application to grant the approved site permissions. The authorizer must provide a reason for each entry; the reasons are typically provided by the VistAWeb user to the local Information Security Officer (ISO) when the user makes the initial request.

The data is saved to a SQL server database table.

The associated authorizer notifies the user and the local ISO of the authorized permissions.

Like the VistAWeb project, the Special User Access project is governed by the same development methodology.

VistA Connections

Compatibility With Patch XWB*1.1*35

As of version 12, VistAWeb is compatible with patch *XWB*1.1*35 – Non Call-back Broker*. This patch allows clients to connect to the VistA broker using a single port/connection, instead of the 2-port/connection method (AKA the "call-back broker") in the old style.

VistAWeb attempts all VistA connections using the newer style defined by patch XWB*1.1*35. If that method fails, it falls-back to the older style of connection.

The consequences of connecting to VistA with the new style is that the client connection could be disconnected by the VistA account after a certain period of inactivity. To remedy this, VistAWeb calls the "broker info" RPC on the VistA system when it firsts connects to get that system's timeout value. Then, VistAWeb sets up an execution thread that runs the "I'm Here" RPC for that VistA connection during inactive times. In the VistAWeb documentation we call this the "ping thread".

VistAWeb runs the "ping thread" in half (1/2) the time indicated as the VistA's timeout value. For example: If the Salt Lake system indicated 180 seconds as the connection timeout, VistAWeb runs the "I'm Here" RPC every 90 seconds if there is no RPC activity on that connection.

Note: *The "ping thread" mechanism is* ***only*** *used for the newer XWB*1.1*35 connections. The older connection method remains unchanged.*

The use of the "ping thread" mechanism allows the user to remain connected to the needed VistA systems until their work is finished, without having to reconnect serially to each VistA system every time a report is requested.

Version 13 introduced a "ping thread" timeout. Since VistAWeb is a web app, sometimes users stop using it without VistAWeb being able to detect this (for example; simply clicking the close "X" on the browser window, or the network connection is interrupted, etc). In order for VistA connections to be able to close properly, the "ping thread" timeout is set for 20 minutes, which is 5 minutes longer than the default timeout value for VistAWeb. After that time, the "ping thread" is terminated, and the VistA connection is allowed to be closed by VistA after its timeout period has elapsed. That way, user's VistA connections will eventually be closed and the port reclaimed for reuse on the server.

Note*: All "ping threads" are terminated and all connections are closed if VistAWeb does detect the user is done (for example, by clicking the "Logoff" button).*

The Appendix D section of this document describes parameters that can be adjusted for the new connection method, how to disable the "ping thread", or how to force VistAWeb to use the new connection method or the old connection method for all VistA connections.

Broker Security Enhancement

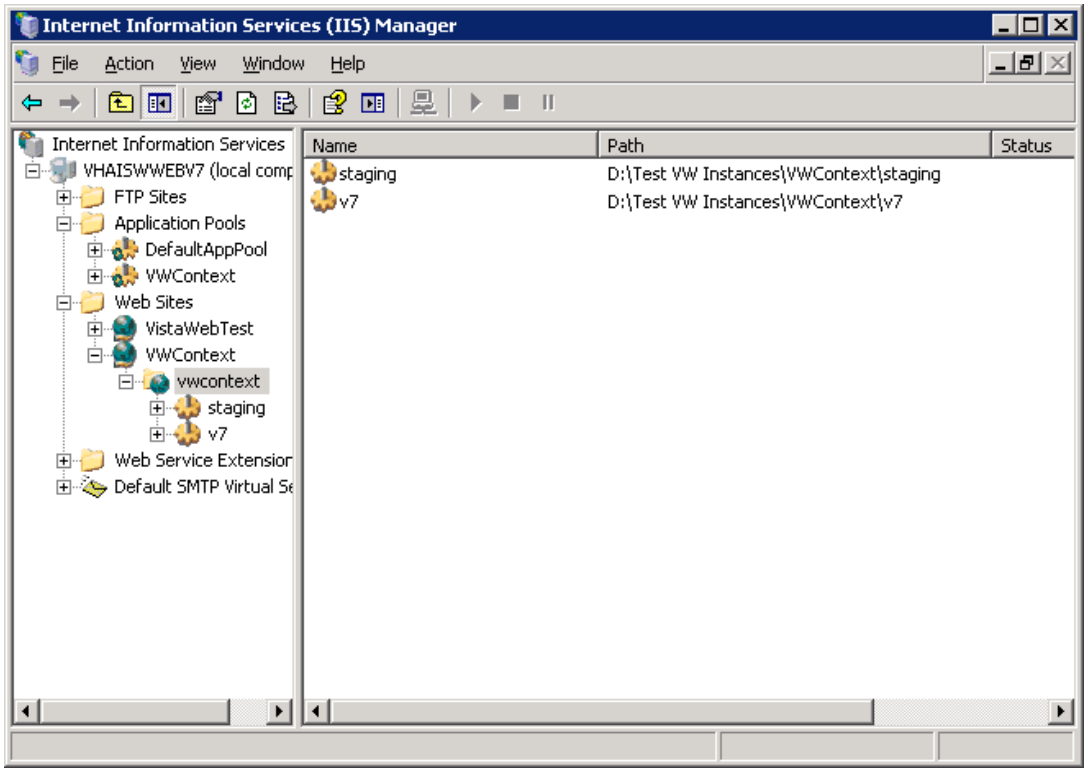
As of version 7, VistAWeb takes advantage of the VistA Broker Security Enhancement (also known as BSE) to set up and authenticate users on "remote" VistA systems. A "remote" VistA system is a VistA account that the user does not have access/verify credentials to access. The Broker Security Enhancement is a mechanism to provide user access to these other systems to provide patient data, which is more secure than the old method of granting access.

In order for this to work, the VistAWeb team has created a web application called the VistAWeb Context service that allows "remote" VistA systems to callback to VistAWeb to validate the remote authorization request. (See the documentation on the BSE from the Common Services group at http://tspr.vista.med.va.gov/warboard/anotebk.asp?proj=994 to see how this functionality works.)

First-Time-Only Setup

1. Back up the SQL Server database. *This step cannot be skipped!*
2. Create a login in SQL Server to allow *only inserting* into the LoggerTable table.
3. Create a web application for the new BSE context service; name it VWContext. Configure the web application to use 10.2.48.10 and unsecured port 12181 (see Figure 5)
    - Specify a new MIME type for this application for "*.do" items
    - This web application will run as an HTTP application (and ***not*** HTTPS).

Figure 5: Configuration of VWContext



4. Run the SQL script entitled "Update CPRS" against the SQL database; this will add a column to one of the tables and populate the column with the data.

Steps to Update in v16 in Staging and SQA - Every Time

1. Remove prior versions of VW and vwContext.
2. Unzip vistaweb_<version>.<date>.zip into the target VW folder.
3. Unzip vistawebcontext_<version>.<date>.zip into the target folder created in step 3 above.
4. Change the <vistaweb>/resources/xml/log4net.xml file to add the login ID and password from step 2 above. Strip out the "Provider=SQLOLEDB" from the connection string.
5. Change the <vistawebcontext>/resources/xml/log4net.xml to add the login ID and password from step 2. Strip out the "Provider=SQLOLEDB" from the connection string.
6. Change the <vistaweb>/web.config file.
   a. Change the vistaConnectionFactory.security phrase to "MY NAME IS SQA01".
   b. Change userActivity.connectionString to specify the SQA SQL Server database (EMR_SQA for SQA and EMR for UAT).
   c. Change version.useFullVersion to "true".
   d. Change allowViewLog to "true".
   e. Change excludeChemHem to "false".
   f. Change the <vistawebcontext>/web.config file to specify the SQA SQL Server database (set it to the same value as what's in VI.b).
   g. Start the vistawebcontext web application, and then start the vistaweb application.

h. Change permissions to image\temp file to allow Network Service to “Write” in vistaweb.
i. Copy vhasites.xml file from existing instance into staging.
j. Test.

SQL Server Database Overview

VistAWeb needs the following tables in order to run and will interact with these tables constantly. The EMR database contains the following tables:

CprsUsers—CPRS-spawned VistAWeb checks this table to see if the CPRS user has logged into the spawned version before. If not, then the user is asked to log into VistAWeb. Once this is done one time, the user's information is added to the CPRSUsers table. Future CPRS-spawned VistAWeb browsers will not ask the user to log in if their information is found in this table.

SpecialUsers—retains the remote site permissions assigned to users.

Log—tracks the movements of users within the VistAWeb application.

Request—tracks remote sites to which users want special user access.

LoggerTable – used by the log4net framework used by VistAWeb to store application errors and other internal logging

BiggerLogger – used by the log4net framework used by VistAWeb to store application messages too big for the *LoggerTable*

UserAuth – used by the Broker Security Enhancement code in VistAWeb and the associated VistAWeb Context service to allow "remote" VistA systems to authenticate back to VistAWeb

Appendix A: Database Schema

- Database Name: EMR
- Database Tables:
  - Log
  - CprsUsers
  - SpecialUser
  - LoggerTable
  - BiggerLogger
  - UserAuth

- Views:
  - LogDesc

Log Creation Script

```
CREATE TABLE [dbo].[Log] (
    [id] [numeric](19, 0) IDENTITY (1, 1) NOT NULL ,
    [requestDate] [datetime] NULL ,
    [remoteAddr] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [userId] [numeric](19, 0) NULL ,
    [userName] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [userSitecode] [varchar] (6) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [userSitename] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [requestPage] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [requestSitecode] [varchar] (6) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [requestSitename] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [patientID] [numeric](19, 0) NULL ,
    [patientName] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [patientSensitivity] [tinyint] NULL ,
    [message] [varchar] (4000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
)
```

Log Index Creation Scripts

```
ALTER TABLE [dbo].[Log] ADD  CONSTRAINT [PK_Log] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (   PAD_INDEX  = OFF,
          STATISTICS_NORECOMPUTE  = OFF,
          SORT_IN_TEMPDB = OFF,
          IGNORE_DUP_KEY = OFF,
          ONLINE = OFF,
          ALLOW_ROW_LOCKS  = ON,
          ALLOW_PAGE_LOCKS  = ON)
 ON [PRIMARY]

CREATE NONCLUSTERED INDEX [requestDate] ON [dbo].[Log]
(
    [requestDate] ASC
)WITH (   PAD_INDEX  = OFF,
          STATISTICS_NORECOMPUTE  = OFF,
          SORT_IN_TEMPDB = OFF,
          IGNORE_DUP_KEY = OFF,
          DROP_EXISTING = OFF,
```

```
          ONLINE = OFF,
          ALLOW_ROW_LOCKS  = ON,
          ALLOW_PAGE_LOCKS  = ON)
ON [PRIMARY]
```

CprsUsers Creation Script

```
CREATE TABLE [dbo].[CprsUsers] (
    [UserID] [numeric](19, 0) IDENTITY(1,1) NOT NULL ,
    [Sitecode] [varchar] (3) NOT NULL ,
    [SiteName] [varchar] (80) ,
    [DUZ] [varchar] (50) NOT NULL ,
    [SSN] [varchar] (9) NOT NULL ,
    [Name] [varchar] (100) NOT NULL ,
   [activeDate] [datetime] NULL ,
   [inactiveDate] [datetime] NULL
)
```

CprsUsers Index Creation Script

```
ALTER TABLE [dbo].[CprsUsers] ADD  CONSTRAINT [PK_CprsUsers] PRIMARY KEY CLUSTERED
(
    [UserID] ASC
)WITH (   PAD_INDEX  = OFF,
          STATISTICS_NORECOMPUTE  = OFF,
          SORT_IN_TEMPDB = OFF,
          IGNORE_DUP_KEY = OFF,
          ONLINE = OFF,
          ALLOW_ROW_LOCKS  = ON,
          ALLOW_PAGE_LOCKS  = ON)
 ON [PRIMARY]
```

SpecialUsers Creation Script

```
CREATE TABLE [dbo].[SpecialUsers] (
    [RecID] [numeric](19, 0) IDENTITY (1, 1) NOT NULL ,
    [UserSiteId] [varchar] (3) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [DUZ] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [UserName] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [Site] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [Reason] [varchar] (200) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [ActiveDate] [datetime] NULL ,
    [DeactiveDate] [datetime] NULL
)
```

SpecialUsers Index Creation Script

```
ALTER TABLE [dbo].[SpecialUsers] ADD  CONSTRAINT [PK_SpecialUsers] PRIMARY KEY
CLUSTERED
(
    [RecID] ASC
)WITH (   PAD_INDEX  = OFF,
          STATISTICS_NORECOMPUTE  = OFF,
          SORT_IN_TEMPDB = OFF,
          IGNORE_DUP_KEY = OFF,
          ONLINE = OFF,
          ALLOW_ROW_LOCKS  = ON,
          ALLOW_PAGE_LOCKS  = ON)
 ON [PRIMARY]
```

LoggerTable Creation Script

```
CREATE TABLE [dbo].[LoggerTable] (
    [Id] [int] IDENTITY (1, 1) NOT NULL,
    [Date] [datetime] NOT NULL,
    [Thread] [varchar] (255) NOT NULL,
    [Level] [varchar] (50) NOT NULL,
    [Logger] [varchar] (255) NOT NULL,
    [Message] [varchar] (4000) NOT NULL,
    [Exception] [varchar] (2000) NULL
)
```

BiggerLogger Creation Script

```
CREATE TABLE [dbo].[BiggerLogger](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [Date] [datetime] NOT NULL,
    [Thread] [varchar](255) NOT NULL,
    [Level] [varchar](50) NOT NULL,
    [Logger] [varchar](255) NOT NULL,
    [Message] [text] NULL,
    [Exception] [varchar](2000) NULL
) ON [DEFAULT] TEXTIMAGE_ON [DEFAULT]
```

UserAuth Creation Script

```
CREATE TABLE [dbo].[UserAuth] (
    [sessionId] [varchar] (80) NOT NULL,
    [sessionType] [varchar] (20) NOT NULL,
    [effectiveDate] [datetime] NOT NULL,
    [inactiveDate] [datetime],
    [status] [varchar] (10),
    [name] [varchar] (100) NOT NULL,
    [userId] [numeric](19, 0) NOT NULL
)
```

UserAuth Index Creation Script

```
CREATE NONCLUSTERED INDEX [UserAuth_sessionUser] ON [dbo].[UserAuth]
(
    [sessionId] ASC,
    [userId] ASC
)WITH (   PAD_INDEX  = OFF,
          STATISTICS_NORECOMPUTE  = OFF,
          SORT_IN_TEMPDB = OFF,
          IGNORE_DUP_KEY = OFF,
          DROP_EXISTING = OFF,
          ONLINE = OFF,
          ALLOW_ROW_LOCKS  = ON,
          ALLOW_PAGE_LOCKS  = ON)
 ON [PRIMARY]
```

Appendix B: VistAWeb Code Samples

Sample HTM (Countdown.htm)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
   <HEAD>
          <title>Countdown</title>
          <link rel="stylesheet" type="text/css"
href="resources/css/main.css"></link>
          <script LANGUAGE="JavaScript">
var now = new Date();
var event = new Date();
event.setMinutes(event.getMinutes() + 2);
var seconds = (event - now) / 1000;
var showSeconds;
ID=window.setTimeout("update();", 1000);

function update() {
   now = new Date();
   seconds = (event - now) / 1000;
   seconds = Math.round(seconds);
   if (seconds > 59) {
          showSeconds = seconds - 60;
   }      else {
          showSeconds = seconds;
          document.Countdown.minutes.value = 0;
   }

   document.Countdown.seconds.value = showSeconds;
   if (seconds == 0)
          returnToCaller("Close");
   else
          ID=window.setTimeout("update();",1000);
}

function returnToCaller(option) {
   window.returnValue = option;
   window.close();
}
          </script>
          <style>
          body
          {
          margin-left: 20px;
          }
          </style>
   </HEAD>
   <body MS_POSITIONING="GridLayout">
          <table border="0" width="100%">
          <tr>
          <td><img src="resources/images/vistaWeb.gif" alt="VistaWeb image"></td>
          <td align="right"><IMG src="resources/images/Timeout.gif" alt="Timeout
image"></td>
          </tr>
          </table>
          <hr color="#cc0000">
          <p>
                 <form name="Countdown" method="post">
                 VistaWeb has not been used for 15 minutes. It will close in the
indicated time
```

```
                    unless you click the "Don't Close" button...
                    <p>
                    <span class="hdr">Time Remaining Until VistaWeb Closes:</span><br>
                    <br>
             <TABLE BORDER="5" CELLSPACING="5" CELLPADDING="0">
             <TR>
             <TD ALIGN="middle" WIDTH="23%"><B>Mins:</B></TD>
             <TD ALIGN="middle" WIDTH="23%"><B>Secs:</B></TD>
             </TR>
             <TR>
             <TD ALIGN="middle"><INPUT type="Text" name="minutes" size="2"
    value="1"></TD>
             <TD ALIGN="middle"><INPUT type="Text" name="seconds" size="2"></TD>
             </TR>
             </TABLE>
                    <br>
             <input type="button" class="myButton"
    onclick="returnToCaller('Continue')" value="Don't close VistaWeb">
             <input type="button" class="myButton" onclick="returnToCaller('Close')"
    value="Close VistaWeb">
                    </form>
        </body>
    </HTML>
```

Sample ASPX Page (TextRecordPage.aspx)

```
<%@ Page language="c#" Inherits="EMR.TextRecordPage" CodeFile="TextRecordPage.aspx.cs" %>
<%@ Register TagPrefix="uc1" TagName="PrintLinkComponent" Src="PrintLinkComponent.ascx" %>
<%@ Register TagPrefix="uc1" TagName="ParamsAndQueryControl" Src="ParamsAndQueryControl.ascx" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
        <HEAD>
                <title></title>
                <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
                <meta name="CODE_LANGUAGE" Content="C#">
                <meta name="vs_defaultClientScript" content="JavaScript">
                <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
                <link href="resources/css/main.css" type="text/css" rel="stylesheet">
                <script language="javascript"
src="resources/scripts/js/functions_lib.js"></script>
                <script language="javascript" src="resources/scripts/js/JSExtras.js"></script>
                <script language="javascript" src="resources/scripts/js/dialogWindow.js"
type="text/javascript"></script>
        </HEAD>
        <body onload="init('TextRpt', '');setBehavior();hideNavigationPopupWarning()">
                <form id="textRecordPageForm" method="post" runat="server">
                        <uc1:ParamsAndQueryControl id="paramsAndQueryControl"
runat="server"></uc1:ParamsAndQueryControl>
                        <asp:TextBox ID="ImageFileName" Runat="server"
CssClass="hideImageText"></asp:TextBox>
                        <asp:TextBox ID="ImageFileAlt" Runat="server"
CssClass="hideImageText"></asp:TextBox><br>
                        <uc1:PrintLinkComponent id="printLinkComponent"
runat="server"></uc1:PrintLinkComponent>
                </form>
                <form name="TimeoutForm" action="Timedout.aspx" target="_top">
                </form>
        </body>
</HTML>
```

Sample Code-Behind Page (TextRecordPage.aspx.cs)

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.Reflection;
using System.Security;
using gov.va.med.vistaweb.util;
using gov.va.med.vistaweb.ui;

namespace EMR
{

    public partial class TextRecordPage : AbstractFlatReport {

            protected void Page_Load(object sender, System.EventArgs e) {
                    base.pageLoad(sender, e);
                    propertyName = Request.QueryString["app"];
                    ImageFileAlt.Text = Request.QueryString["alt"];
                    ImageFileName.Text = propertyName;
                    if (Request.QueryString["dr"]==null ||
!"y".Equals(Request.QueryString["dr"].ToLower())) {
                            if (ParamAndQueryComponent!=null) ParamAndQueryComponent.Visible =
false;
                    }
            }

            protected override IPropertyParameters getQueryParameters() {
                    if (ParamAndQueryComponent!=null && !ParamAndQueryComponent.Visible) {
                            return null;
                    }
                    return base.getQueryParameters();
            }

            protected override gov.va.med.vistaweb.ui.IParamAndQueryComponent
ParamAndQueryComponent {
                    get { return paramsAndQueryControl; }
            }

            protected override gov.va.med.vistaweb.ui.IPrintLinkComponent PrintLinkComponent {
                    get { return printLinkComponent; }
            }

            protected override gov.va.med.vistaweb.ui.IWaitOrCancelComponent
WaitOrCancelComponent {
                    get { return null; }
            }

            #region Web Form Designer generated code
            override protected void OnInit(EventArgs e)
            {
                    //
                    // CODEGEN: This call is required by the ASP.NET Web Form Designer.
                    //
                    InitializeComponent();
                    base.OnInit(e);
            }

            /// <summary>
            /// Required method for Designer support - do not modify
            /// the contents of this method with the code editor.
            /// </summary>
            private void InitializeComponent()
```

```
            {
                this.PreRender += new System.EventHandler(this.preRender);

            }
            #endregion

    }
}
```

Appendix C: Security Guide

Security for VistAWeb is handled at the local VistA level and at the national VistAWeb server level. A Broker Security Enhancement (BSE) patch enables broker security, while a VistAWeb patch identifies the VistA site as a legitimate user of the national VistAWeb application.

When users log into their VistA/CPRS accounts, they are first authenticated by VistA/CPRS through the use of a valid access and verify code pair. Then when the users launch VistAWeb, VistAWeb verifies that the user is an authenticated user by sending an authentication message to the VistA system via the broker. In the standalone mode of operation, the users direct their Internet Explorer browsers to the VistAWeb Home Page. When they select and log into their local site or one of the sites to which they have been granted access through the Special User Access program, VistA user verification occurs and then the VistAWeb site authentication and notification process takes place.

Apart from having installed the necessary patches for BSE and VistAWeb, nothing is required from the local Information Resource Management (IRM) staff or Information Security Officer (ISO).

Appendix C: Configurable Items

The use of the Spring Framework enables a number of items to be configurable in VistAWeb. They are listed below in no particular order. If you change any of these parameters, they will only take effect after a restart of the web server (the IIS service).

1. VistA Connection Parameters

   In the `resources/contexts/`**`connectionContext.xml`** file there are several parameters that can be adjusted.

   Note: Version 12 introduced a new VistA connection class called *VistAConnection2*, which is compatible with Broker Patch XWB*1.1*35 (see the Compatibility With Patch XWB*1.1*35 section of this document). Version 12 also added a VistA connection fallback façade, which allows a specified VistA connection class to be tried, and then to fallback to a different connection class if the first one fails. Thus, all VistA connections in version 12 are set up to attempt to connect with the *VistaConnection2* class first, then the system resorts to the *VistaConnection* class if that fails.

   a. Connection Try Limit – limits how many times a connection failure will be tolerated before throwing an error. Change the *`connectionTryLimit`* in *`abstractVistaConnection`* for the affect all VistA connections, or change it in *`abstractVistaConnection1`* to only affect *`VistaConnection`* class instances.
   b. Initial Timeout – how many milliseconds to wait for the TCP connection open to complete successfully (the TCP/IP SYN – SYN/ACK handshake sequence). Changing *`initialTimeout`* in *`abstractVistaConnection1`* changes how the *`VistaConnection`* class behaves and changing *`initialTimeout`* in *`abstractVistaConnection2`* changes how the *`VistaConnection2`* class behaves.
   c. Timeout – how long to wait for a response from VistA on an RPC call before throwing an error. Changing *`timeout`* in *`abstractVistaConnection1`* changes how the *`VistaConnection`* class behaves and changing *`timeout`* in *`abstractVistaConnection2`* changes how the *`VistaConnection2`* class behaves.
   d. "Pinging" – refers to the *VistaConnection2* class's behavior in regards to calling the "`XWB IM HERE`" RPC. This RPC is only used for *VistaConnection2* connections. In the *`abstractVistaConnection2`* entry, you can change the following items:
      i. `IsUsingLocalPingInterval`
         1. If set to "`true`" will try to get the VistA connection timeout value from a call to the "`XWB GET BROKER INFO`" RPC. If the RPC succeeds, then the "`XWB IM HERE`" RPC will be called in half

the time indicated if there is no activity on the connection. For example: if the "XWB GET BROKER INFO" RPC returns 180, then the connection will call the "XWB IM HERE" RPC every 90 seconds if no other RPC has been called during that time. If the "XWB GET BROKER INFO" RPC fails, then the value in *DefaultPingIntervalSecs* will be used as the interval.

2. If set to "false", will always use the interval in *DefaultPingIntervalSecs*

ii. DefaultPingIntervalSecs – the interval to wait for connection activity before the "XWB IM HERE" RPC is called.

NOTE: If *IsUsingLocalPingInterval* is set to "false" and *DefaultPingIntervalSecs* set to 0 or less, the "XWB IM HERE" RPC will never be used or called.

iii. MaxPingMinutes – the number of minutes that "pinging" will function before quitting. *It is always important to set this to a value or some abandoned VistA connections may never get closed if "pinging" is used!*

e. Socket Linger – the wait time for a connection being closed. Change the *socketLinger* to the number of seconds the execution thread will wait for a connection being closed. Normally this is set to 0.

2. VistA Behavior

In the resources/contexts/**daoContext.xml** file there are several parameters that can be adjusted:

a. VistA Connection – which VistA connection object to use. If you wish to disable the fallback façade and use either *VistaConnection* (older style broker connections) or *VistaConnection2* (newer style broker connections) for all VistA connections, you can change the *vistaConnection* value in *protocol_VISTA* to "vistaConnection" or "vistaConnection2", respectively.

b. Site ID Ignore List – configuration parameters to ignore certain entries from the MPI. The following properties are applied in the order given here:

i. SiteIdIgnoreList – which station numbers to ignore. If the value begins with an "at" symbol (@), then anything that follows that symbol will be matched as a regular expression and ignored if matched. Otherwise, entries with that exact station number will be ignored.

ii. remoteExcludeTypes – the type of MPI entries to ignore. Currently VistAWeb ignores entries with type "other".

iii. remoteKeepPatterns – a list station numbers, where if matched, will be kept no matter what the prior 2 sections has excluded. The station

numbers are matched using a *starting* match. For example, if this field specified "200N", any station number starting with "200N" would be matched and kept (such as 200NKP, 200NDOD, etc).

c. Check Details Message – the message that gets displayed in the grid for certain reports if there is an error.
d. MPI Error Tollerance – error tolerance can be turned off or on, and if turned on, you can specify what errors to ignore
    i. IsMpiErrorTolerant – set "true" to ignore all errors, "false" to only ignore errors specified in the "toleratedMpiErrors" property
    ii. toleratedMpiErrors – the exact error text from VistA to match to ignore. If any other error text is found, then VistAWeb will display an error to the user.
e. contextString – the default menu context to use for every user for all VistA connections. However, if a permission attribute is set for a VistA site in the sites file (VhaSites.xml), the permission attribute is used instead. In the case of remote VistA systems, this same menu option will need to be specified in the REMOTE APPLICATION file (see the BSE documentation for details).

3. Usage Stats
    a. In the resources/context/**daoContext.xml** file, in the statsDao object, you can change the following setting which will affect how VistAWeb reports usage statistics:
        i. SqlGlobalStats: you can specify which pages to exclude for the global statistics
        ii. SqlSiteStats: you can specify which pages to exclude for the site statistics

4. Removing Reports From the Reports Menu
    a. In the resources/context/**mainContext.xml** file, in the object with attribute type="TreeTOC.aspx", you can comment out any report or sections you don't want to make available to the users.

5. Application Storage
    a. In the resources/context/**mainContext.xml** file, in the appStorage_connectionFactory object, you can define what kind of database that VistAWeb uses as its backing data store. Read the notes in this file for instructions on configurations.

6. Report Changes
    a. In the `resources/context/`**`pageContext.xml`** file, you can specify how a grid column behaves.
        i. `defaultSortedColumnIndex`: the column which is the default sorted column (0 based indexing)
        ii. `sortExpression`: the column(s) used for sorting that particular column when the user selects that column to be sorted. You must use the column property names, separated by a comma. You may optionally specify ascending (with the **`asc`** keyword) or descending (with the **`desc`** keyword). Read through the `pageContext.xml` file for examples.
        iii. `headerTitle`: these attributes contain what is displayed as the column header text
    b. In the `resources/context/`**`pageContext.xml`** file, you can remove columns from grid reports
        i. Anywhere you see an element with an attribute like **`type="gov.va.med.vistaweb.ui.grid.GridColumn,emr.cs"`**, you can comment out this element and the column will no longer show up in the report.

7. CDS 2 / HDR 2
    a. CDS 2 is the interface to the HDR 2 data source. The configuration file for this data source is called `resources/xml/caipConfig_production.xml`. Parameters like "`siteName`", "`application`", and "`NDSconnection/URL`" can be specified with this file.
    b. The files in the `resources/xml/xsl/cds` folder are the schema files for the data retrieved from CDS 2.1.*.

Appendix D: System Information Reports

Information for the production VistAWeb servers can be found at the following links:

Web Server

http://vaww.sms.aac.va.gov/SMSReporting/MachDetails.asp?Machine=vaausnvwweb200

SQL Server

http://vaww.sms.aac.va.gov/SMSReporting/MachDetails.asp?Machine=VAAUSNVWSQL200

Appendix E: RPCs Used by VistAWeb

DG SENSITIVE RECORD ACCESS

DG SENSITIVE RECORD BULLETIN

ORPRF GETFLG

ORPRF HASFLG

ORQOR DETAIL

ORQQAL DETAIL

ORQQCN DETAIL

ORQQCN LIST

ORQQCN MED RESULTS

ORQQPL PROBLEM LIST

ORQQVI1 GRID

ORVW FACLIST

ORWCIRN FACLIST

ORWCV DTLVST

ORWCV VST

ORWLR CUMULATIVE REPORT

ORWLRR INTERIM

ORWLRR MICRO

ORWORB FASTUSER

ORWPS COVER

ORWPS DETAIL

ORWPS MEDHIST

ORWPT DIEDON

ORWPT FULLSSN

ORWPT ID INFO

ORWPT PTINQ

ORWPT SELECT

ORWPT1 PCDETAIL

ORWPT1 PRCARE

ORWRA IMAGING EXAMS1

ORWRP REPORT LISTS

ORWRP REPORT TEXT

ORWRP2 HS COMP FILES

ORWRP2 HS COMPONENT SUBS

ORWRP2 HS COMPONENTS

ORWRP2 HS FILE LOOKUP

ORWRP2 HS REPORT TEXT

ORWSR RPTLIST

ORWU NEWPERS

ORWU USERINFO

ORWU VERSRV

TIU DOCUMENTS BY CONTEXT

TIU GET LINKED PRF NOTES

TIU GET PRF TITLE

TIU GET RECORD TEXT

XUS AV CODE

XUS CVC

XUS INTRO MSG

XUS SET VISITOR

XUS SIGNON SETUP

XUS SIGNON SETUP

XWB CREATE CONTEXT

XWB GET VARIABLE VALUE

Appendix F: Configuring a build to work on a computer that doesn't have Visual Studio 2010

To get a build to work on a computer that doesn't have Visual Studio 2010 installed, follow the solution that's outlined in the blog: http://blogs.msdn.com/b/webdevtools/archive/2010/05/26/visual-studio-2010-web-deployment-projects-rtw-available-now.aspx.

Here are the steps to take in order for the build to work. Note, that "\\name" represents a development computer where VS2010 is installed:

1. Install Visual Studio 2010 Web Deployment Project – RTW from http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=24509.

2. Install Windows SDK 7 and .NET framework 4 from this link http://msdn.microsoft.com/en-us/windows/bb980924

3. Copy \\Name\Program Files\Microsoft SDKs\Windows\v7.0A\bin\aspnet_merge.exe to C:\Program Files\MSBuild\Microsoft\WebDeployment\v10.0\aspnet_merge.exe

4. Edit C:\Program Files\MSBuild\Microsoft\WebDeployment\v10.0\Microsoft.WebDeployment.targets file by adding two lines highlighted in yellow:
   from:
   ```
   <!--*******************************************************-->
   <!--Include the WPP targets file-->
   <Import Project="..\..\VisualStudio\v10.0\Web\Microsoft.Web.Publishing.targets"
   Condition="$(_WDP_WPPExists)" />
   <!--*******************************************************-->
   ```
   to:
   ```
   <!--*******************************************************-->
   <!--Include the WPP targets file-->
   <Import Project="..\..\VisualStudio\v10.0\Web\Microsoft.Web.Publishing.targets"
   Condition="$(_WDP_WPPExists)" />
   <Import Project="..\..\VisualStudio\v10.0\Web\Microsoft.Web.Publishing.targets" />
   <Import Project="..\..\VisualStudio\v10.0\WebApplications\Microsoft.WebApplication.targets"
   />
   <!--*******************************************************-->
   ```
me

5. Copy \\Name\Program Files\MSBuild\Microsoft\VisualStudio\v10.0\Web and \\Name\Program Files\MSBuild\Microsoft\VisualStudio\v10.0\WebApplications directories to C:\Program Files\MSBuild\Microsoft\VisualStudio\v10.0.