KERNEL AUTHENTICATION & AUTHORIZATION FOR J2EE (KAAJEE)

DEPLOYMENT GUIDE

KAAJEE Version 1.0.1.xxx &
SSPI Version 1.0.0.010

April 2009

Department of Veterans Affairs (VA)
Office of Information & Technology (OI&T)
Office of Enterprise Development (OED)

Revision History

Documentation Revisions

The following table displays the revision history for this manual. Revisions to the documentation are based on patches and new versions released to the field.

Table i. Documentation revision history

Date	Revision	Description	Author(s)
04/02/09	1.0	Initial KAAJEE 1.0.1.xxx documentation. **KAAJEE 1.0.1.003** **SSPI 1.0.0.010** **Kernel Patch XU*8.0*451**	KAAJEE Development Team, Oakland, CA Oakland Office of Information Field Office (OIFO): • Development Manager—Jack Schram • Developer—Alan Chan • SQA—Gurbir Singh • Technical Writer—Thom Blom

Patch Revisions

For a complete list of patches related to this software, please refer to the Patch Module on FORUM.

NOTE: Kernel is the designated custodial software application for KAAJEE; however, KAAJEE comprises multiple patches and software releases from several Health***e***Vet-VistA applications.

REF: For the specific KAAJEE software and VistA M Server patches required for the implementation of KAAJEE, please refer to Table 1-2 in the "KAAJEE" topic in Chapter 1 in this manual.

Contents

Figures and Tables

Figures

Tables

Acknowledgements

The Kernel Authentication and Authorization Java (2) Enterprise Edition (KAAJEE) development team consists of the following Office of Enterprise Development (OED) personnel (listed alphabetically within a category):

- Program Manager—Catherine Pfeil
- Development Manager—Jack Schram
- Developers—Alan Chan and Jose Garcia
- Software Quality Assurance (SQA)—Gurbir Singh
- Technical Writer—Thom Blom

The KAAJEE development team would like to thank the following sites/organizations/personnel for their assistance in reviewing and/or testing KAAJEE-related software and documentation (project development teams are listed alphabetically):

- Emergency Department Integration System (EDIS)—Development Team
- Spinal Cord Dysfunction (SCD)

Orientation

This manual is intended for use in conjunction with the deployment of the Kernel Authorization and Authentication for J2EE (KAAJEE) software. It outlines the details of KAAJEE-related software and gives guidelines on how the software is used within Health***e***Vet-Veterans Health Information Systems and Technology Architecture (VistA).

The intended audience of this manual is all key stakeholders. The primary stakeholder is the Office of Enterprise development (OED). Additional stakeholders include:

- Health***e***Vet-VistA application developers of Web-based applications in the WebLogic 8.1 (SP4 or higher) Application Server environment.
- Information Resource Management (IRM) and Information Security Officers (ISOs) at Veterans Affairs Medical Centers (VAMCs) responsible for computer management and system security.
- Enterprise Product Support (EPS).
- VAMC personnel who will be using Health***e***Vet-VistA Web-based applications running in the WebLogic 8.1 (SP4 or higher) Application Server environment.
-
- There are no special legal requirements involved in the use of KAAJEE-related software.

How to Use this Manual

This manual is divided into three major parts:

- User Guide—Provides general overview of the KAAJEE software.
- Developers Guide—Provides step-by-step instructions for Health***e***Vet-VistA developers to follow and Application Program Interfaces (APIs) to use when writing Web-based applications incorporating the KAAJEE authorization and authentication functionality.
- Systems Management Guide—Provides implementation, maintenance, and security overview for IRM and ISO personnel.

Throughout this manual, advice and instructions are offered regarding the use of KAAJEE software and the functionality it provides for Health***e***Vet-Veterans Health Information Systems and Technology Architecture (VistA) software products.

This manual uses several methods to highlight different aspects of the material:

- Various symbols/terms are used throughout the documentation to alert the reader to special information. The following table gives a description of each of these symbols/terms:

Table ii. Documentation symbol/term descriptions

Symbol	Description
	**NOTE/REF:** Used to inform the reader of general information including references to additional reading material.
	**CAUTION or DISCLAIMER:** Used to inform the reader to take special notice of critical information.

- Descriptive text is presented in a proportional font (as represented by this font).
- "Snapshots" of computer online displays (i.e., roll-and-scroll screen captures/dialogues) and computer source code, if any, are shown in a *non*-proportional font and enclosed within a box.
  - User's responses to online prompts and some software code reserved/key words will be bold typeface type.
  - Author's comments, if any, are displayed in italics or as "callout" boxes.

    NOTE: Callout boxes refer to labels or descriptions usually enclosed within a box, which point to specific areas of a displayed image.

- Java software code, variables, and file/folder names can be written in lower or mixed case.
- All uppercase is reserved for the representation of M code, variable names, or the formal name of options, field/file names, and security keys (e.g., the XUPROGMODE key).

Assumptions about the Reader

This manual is written with the assumption that the reader is familiar with the following:

- VistALink—VistA M Server and Application Server software
- Linux (i.e., Red Hat Enterprise ES 3.0 or higher) or Microsoft Windows environment
- WebLogic 8.1 (SP4 or higher)—Application Server
- Oracle 9i—Database
- Health***e***Vet-VistA computing environment
- Java Programming language:
  - Java Integrated Development Environment (IDE)
  - J2SE™ Development Kit (JDK)
  - Java Authentication and Authorization Services (JAAS) programming

- M programming language (i.e., Kernel Patch XU*8.0*451)

This manual provides an overall explanation of the installation procedures and functionality provided by the VistA Automated Access Request software; however, no attempt is made to explain how the overall Health*e*Vet-VistA programming system is integrated and maintained. Such methods and procedures are documented elsewhere. We suggest you look at the various VA home pages on the World Wide Web (WWW) and VA Intranet for a general orientation to Health*e*Vet-VistA. For example, go to the Department of Veterans Affairs (VA) Office of Information and Technology (OI&T) VistA Development Intranet Website:

http://vista.med.va.gov

Reference Materials

Readers who wish to learn more about KAAJEE should consult the following:

- *Kernel Authentication & Authorization for J2EE (KAAJEE) Installation Guide*
- *Kernel Authentication & Authorization for J2EE (KAAJEE) Deployment Guide*, this manual
- KAAJEE Website: http://vista.med.va.gov/kernel/kaajee/index.asp
- *Kernel Systems Management Guide*
- *VistALink Installation Guide*
- *VistALink System Management Guide*
- *VistALink Developer Guide*

REF: For more information on VistALink, please refer to the VistALink documentation located on the VHA Software Document Library (VDL) Website at the following Website:

http://www.va.gov/vdl/application.asp?appid=163

Health*e*Vet-VistA documentation is made available online in Microsoft Word format and Adobe Acrobat Portable Document Format (PDF). The PDF documents *must* be read using the Adobe Acrobat Reader, which is freely distributed by Adobe Systems Incorporated at the following Website:

http://www.adobe.com/

Health*e*Vet-VistA documentation can be downloaded from the VHA Software Document Library (VDL) Website:

http://www.va.gov/vdl/

HealthVet-VistA documentation and software can also be downloaded from the Enterprise Product Support (EPS) anonymous directories:

- Preferred Method download.vista.med.va.gov

  This method transmits the files from the first available FTP server.

- Albany OIFO ftp.fo-albany.med.va.gov
- Hines OIFO ftp.fo-hines.med.va.gov
- Salt Lake City OIFO ftp.fo-slc.med.va.gov

DISCLAIMER: The appearance of any external hyperlink references in this manual does not constitute endorsement by the Department of Veterans Affairs (VA) of this Website or the information, products, or services contained therein. The VA does not exercise any editorial control over the information you may find at these locations. Such links are provided and are consistent with the stated purpose of this VA Intranet Service.

I. User Guide

This is the User Guide section of this supplemental documentation for Kernel Authentication and Authorization Java (2) Enterprise Edition (KAAJEE). It is intended for use in conjunction with the KAAJEE software. It details the user-related KAAJEE documentation (e.g., overview of the KAAJEE sub-project), management of KAAJEE-related software, etc.).

1. KAAJEE Overview

Introduction

The Kernel Authentication and Authorization for Java (2) Enterprise Edition (KAAJEE) software was developed by the Office of Enterprise development (OED).

Kernel is the designated custodial software application for KAAJEE; however, KAAJEE comprises multiple software and patches from several Health***e***Vet-VistA applications.

REF: For the specific KAAJEE software and VistA M Server patches required for the implementation of KAAJEE, please refer to Table 1-2 in the "KAAJEE" topic in this chapter.

KAAJEE addresses the Authentication and Authorization (AA) needs of Health***e***Vet-VistA Web-based applications in the J2EE environment. Over the long term, the Department of Veterans Affairs (VA) will provide AA services to perform end-user Authentication and Authorization enterprisewide; however, in the interim period, OED has a choice to make as to which AA mechanism(s) would be the most effective. This applies both to the needs of the applications themselves, as well as in anticipation of an expected migration to the future AA solution.

Most major J2EE application servers (e.g., WebLogic 8.1 [SP4 or higher] and Oracle 9iAS [or higher]) allow enterprises to override the default source of AA and replace it with custom, enterprise-specific sources for AA.

KAAJEE authenticates against a VistA M Server first with Access and Verify codes via VistALink's AV connection spec (i.e., KaajeeVistaLinkConnectionSpec). After the user has been properly authenticated against a VistA M Server, KAAJEE dynamically creates a temporary username and password and populates this into a Structured Query Language (SQL) database via custom Security Service Provider Interfaces (SSPIs). This username and password is needed for the second level/phase/pass authentication for the J2EE container.

REF: For more information on SSPIs and the overall KAAJEE-related AA process please refer to the "Security Service Provider Interfaces (SSPI)" topic and or Figure 1-1 in this chapter.

Currently, Kernel maintains the primary VistA and Health***e***Vet-VistA user store (i.e., NEW PERSON file [#200]), which provides both Authentication and Authorization (AA) services for all VistA and Health***e***Vet-VistA applications. By leveraging Kernel, KAAJEE authenticates and authorizes J2EE Web users by using Kernel's AA capabilities.

Some potential advantages to employing Kernel as the AA source include the following:

- Provides a single point of user management for existing and new Health***e***Vet-VistA applications.
- Allows the use of an existing credential—the Access and Verify code—for Authentication and Authorization, rather than introducing a new security credential.
- Eliminates the need to maintain a mapping from WebLogic accounts to VistA M Server Kernel accounts.
- Avoids an additional user store, which simplifies the migration to the future AA solution.

- Partitions user authorizations by Veterans Health Administration (VHA) site.

Some potential KAAJEE strategy limitations due to employing Kernel as the AA source include the following:

- Kernel user accounts are not currently VA-wide; instead, they are facility-specific.
- Users *must* have an active VistA M Server Kernel account on some VistA system. Not all users fit this requirement (e.g., Veterans Affairs Central Office [VACO] users).
- This strategy introduces a dependency on the M system's availability, to perform virtually any function in a J2EE application.
- Correlating a user at one VA facility with the same user at a different VA facility is not supported, given the current lack of an enterprise-wide VA person identifier (e.g., VA-wide Person Identifier [VPID]).

REF: KAAJEE does *not* currently use the Department of Veterans Affairs Personal Identification (VPID), since this field is not currently populated enterprise-wide.

The KAAJEE software provides a Kernel-based Authentication and Authorization (AA) service for all Health***e***Vet-VistA Web-based applications in the J2EE/WebLogic environment.

Currently, there are several Health***e***Vet-VistA J2EE Web-based applications released (e.g., Blind Rehab, Patient Advocate Tracking System [PATS], and Veterans Personal Finance System [VPFS]). KAAJEE provides the Authentication and Authorization (AA) services to these applications.

KAAJEE is designed to run on the WebLogic 8.1 (SP4 or higher).

This manual discusses in more detail the major software modules that, together, provide for KAAJEE functionality and how to deploy KAAJEE-enabled J2EE Form-based Authentication framework and the Security Service Provider Interfaces (SSPIs).

Table 1-1. Dependencies—KAAJEE-related software applications/modules

Module	Description
WebLogic 8.1 (SP4 or higher) Application Server (running)	A WebLogic 8.1 (SP4 or higher) server uses security provider packages that allow a J2EE application running in WebLogic 8.1 (SP4 or higher) to draw its Authentication and Authorization from Kernel on the VistA M Server. **NOTE:** A J2EE standard for pluggable authentication for J2EE servers is underway[1], but won't be finalized until J2EE 1.5.
VistALink 1.5 (fully patched)	The Application Server *must* also have the VistALink software deployed and running. VistALink provides connectivity between KAAJEE and the VistA M Server.

[1] JSR-196, http://www.jcp.org/en/jsr/detail?id=196.

Module	Description
Standard Data Services (SDS) 3.0 (or higher)	KAAJEE makes internal API calls to the SDS Database/Tables. **NOTE:** KAAJEE works with SDS 3.0 or higher; however, KAAJEE 1.0.1.xxx distributes SDS 13.0 client jar files as part of the Sample Web Application. If you deploy the both the KAAJEE Sample Web Application and your own Web-based application on the same WebLogic Application Server domain instance and intend to use a different version of SDS, those client jar files will need to be swapped out for the appropriate version of the SDS client jar files. Otherwise, There may be a conflict if both applications reference the same JNDI tree.

Features

KAAJEE provides the following high-level features and functionality:

- Prompts users to enter their Access and Verify code when he/she attempts to access a protected application resource for the first time during a user session.
- Validates the entered Access and Verify code against the M system/division selected by the user at logon.
- Permits administrators to configure the display list of M systems, by division, against which an end-user can log in.
- Returns all VistA M Server J2EE security keys and uses these as the basis for authorization decisions, as each security key is cached as a WebLogic group name. The KAAJEE SSPIs currently use an external Oracle 9i database to store this information for later authentication (see Figure 1-1).

  KAAJEE roles are defined by the list of roles in the web.xml file, VistA M Server J2EE security keys, and WebLogic group names found in your application's weblogic.xml file.

  REF: For more information on groups and roles, please refer to Chapter 5, "Role Design/Setup/Administration," in this manual.

- (optional) Maps J2EE security role names with security key role names. Through <security-role-assignment> tags (e.g., in weblogic.xml) the actual J2EE security role names can be different than the security key role names. This mapping is optional, because if the same names are used throughout, no <security-role-assignment> tags are required.

  REF: For a sample spreadsheet showing a mapping between WebLogic group names (i.e., principals) with J2EE security role names, please refer to "Appendix B—Mapping WebLogic Group Names with J2EE Security Role Names" in this manual.

- Transforms valid Access and Verify codes into a J2EE-compatible username (e.g., "kaaj_DUZ_8888~CMPSYS_523") and password, and submits the information to the J2EE

container. It then passes the submitted information to the KAAJEE SSPIs, which validate the username and makes that username the current user.

Application developers can use the HttpServletRequest.getRemoteUser servlet method to return demographic data, such as the KAAJEE-created username (e.g., "kaaj_DUZ_8888~CMPSYS_523").

REF: For more information on formatting J2EE usernames, please refer to the "J2EE Username Format" topic in Chapter 7, "Programming Guidelines," in this manual.

- Calls the KAAJEE SSPIs when the J2EE container checks user roles, which checks the role cache for the given user created at user login. This allows user authorizations to be managed on the VistA M Server, and yet have fast response time in the J2EE application.
- Provides user demographics information, which includes the selected Division at login, user DUZ, and user Name, all which are available to the application after login via the Session object (cookie).

REF: For more information on the user demographics provided, please refer to the following:

- "LoginUserInfoVO Object" topic in Chapter 7, "Programming Guidelines," in this manual.
- VistALink 1.5 documentation. The VistALink documentation is located on the VHA Software Document Library (VDL) Website at the following Website:

  http://www.va.gov/vdl/application.asp?appid=163

- Uses the SIGN-ON LOG file (#3.081) on the VistA M Server (i.e., the same M system used for user authentication) to track user logons and logoffs.

REF: For more information on the SIGN-ON LOG file (#3.081), please refer to the *Kernel Systems Management Guide*.

J2EE container-managed enforcement of security, both programmatic and declarative, is fully enabled with KAAJEE.

Deployment of KAAJEE for a given J2EE application requires the KAAJEE components to be integrated with the application, because the J2EE servlet specification requires J2EE Form-based Authentication to run within the scope of the application using it.

KAAJEE Software Dependencies for Consuming Applications

Kernel is the designated custodial software application of the KAAJEE-related software; however, KAAJEE comprises/depends on multiple patches/software releases from several Health*e*Vet-VistA applications, as follows (listed by category):

Table 1-2. Dependencies—KAAJEE software dependencies for consuming applications

Category	Software	Version	Patch/ Software Release	Subject/Description
Application Server	KAAJEE	1.0.1.xxx	1.0.1.xxx	Updated KAAJEE server software, to be released with Kernel Patch XU*8.0*451.
	SSPIs	1.0.0.010	1.0.0.010	KAAJEE SSPIs server software.
	VistALink	1.5 (fully patched)	XOBS 1.5	Updated VistALink server software.
	Kernel	8.0 (fully patched)	XU*8.0*451	KAAJEE Login Page—Removal of Refresh Button. This patch will be released with KAAJEE 1.0.1.xxx. This patch provides the following functionality or bug fixes: • Enhanced Login Functionality: – Removed Refresh button from KAAJEE login page. – Added JavaScript code for client-side sorting of Institutions. – Provided Access code ; Verify code capability in one line. – Added support for parameter passing of Default Institution and Institution sorting preferences. This addresses the issues of persistent cookies when using Thin Clients and Terminal Servers. – Made the KAAJEE Login Web page more Section 508 friendlier. • Added Sample Web Application—Provide KAAJEE Sample Web Application. • Updated Software Version Support: – Compiled and tested KAAJEE against SDS 13.0. – Compiled and tested KAAJEE against VistALink 1.5.1.xxx. • Bug Fixes: – Fixed issue with KAAJEE login not updating LAST SIGN-ON DATE/TIME field (#202) in the NEW PERSON file (#200).

Category	Software	Version	Patch/ Software Release	Subject/Description
				– Fixed Response already committed error—The code that was fixed was associated with processing the persistent cookie information on the Application Server. This fix should also fix the extra M process that was created.

REF: For specific VistA M Server patch details, please refer to the Patch Module on FORUM.

REF: For a list of the Commercial-Off-The-Shelf (COTS) software required for KAAJEE, please refer to Table 8-6 in Chapter 8, "Implementation and Maintenance (J2EE Site)," in this manual.

Security Service Provider Interfaces (SSPI)

The Security Service Provider Interfaces (SSPIs) can be used by developers and third-party vendors to develop security providers for the WebLogic Server environment. SSPIs allow customers to use custom security providers for securing WebLogic Server resources. [2]

Security providers are modules that "plug into" a WebLogic Server security realm to provide security services to applications. They call into the WebLogic Security Framework on behalf of applications implementing the appropriate SSPIs from the weblogic.security.spi package to create runtime classes for the security provider.[3]

Some of the WebLogic security providers and utilities include (descriptions taken from WebLogic Website):

- WebLogic Authentication Provider—"Supports delegated username/password authentication, and utilizes an embedded LDAP server to store, edit, and list user and group information." [4]

  NOTE: KAAJEE (Iteration 1) does *not* use WebLogic's embedded LDAP server. It uses an Oracle 9i database to store users and groups by using SSPIs (see Figure 1-1).

- WebLogic Identity Assertion Provider—"Supports certificate authentication using X.509 certificates." [5]
- WebLogic Principal Validation Provider—"Signs and verifies the authenticity of a specific type of principal, much as an Identity Assertion provider supports a specific type of token; therefore,

[2] WebLogic Website, http://e-docs.bea.com/wls/docs81/secintro/1055098
[3] WebLogic Website, http://e-docs.bea.com/wls/docs81/secintro/1055098
[4] WebLogic Website, http://e-docs.bea.com/wls/docs81/secintro/archtect.html#1044113
[5] WebLogic Website, http://e-docs.bea.com/wls/docs81/secintro/archtect.html#1040282

you can use the WebLogic Principal Validation provider to sign and verify principals that represent WebLogic Server users or WebLogic Server groups." [6]

- WebLogic Authorization Provider—"Supplies the default enforcement of authorization for this version of WebLogic Server. Using a policy-based authorization engine, the WebLogic Authorization provider returns an access decision to determine if a particular user is allowed access to a protected WebLogic resource." [7]
- WebLogic Role Mapping Provider—"Determines dynamic roles for a specific user (subject) with respect to a specific protected WebLogic resource for each of the default users and WebLogic resources." [8]
- WebLogic Auditing Provider—"Records information from a number of security requests, which are determined internally by the WebLogic Security Framework. The WebLogic Auditing provider also records the event data associated with these security requests, and the outcome of the requests." [9]
- WebLogic MBeanMaker Utility—This command-line utility takes an MBean Definition File (MDF) as input and output files to generate an MBean type, which is used to configure and manage the security provider.[10]

REF: For more information on the WebLogic security providers, utilities, and other related information, please visit the following WebLogic Websites:

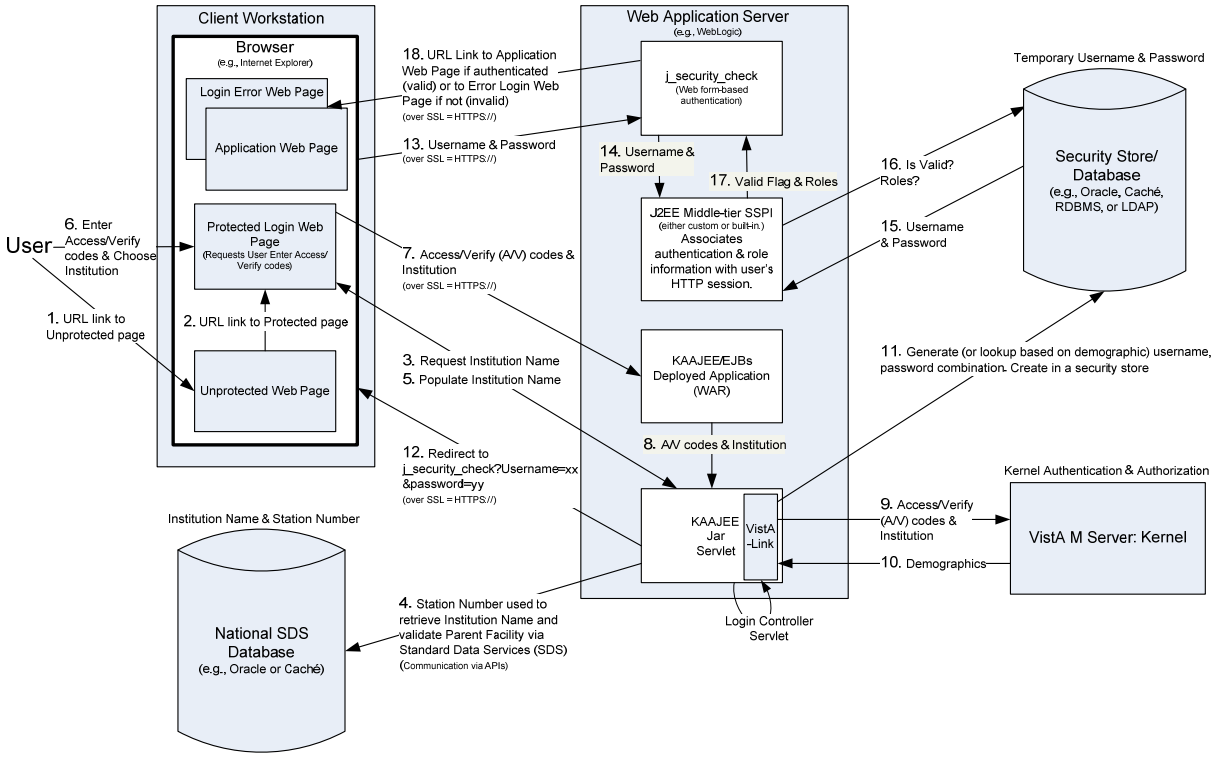
http://e-docs.bea.com/wls/docs81/secintro/archtect.html

http://e-docs.bea.com/wls/docs81/secintro/terms.html

[6] WebLogic Website, http://e-docs.bea.com/wls/docs81/secintro/archtect.html#1049504
[7] WebLogic Website, http://e-docs.bea.com/wls/docs81/secintro/archtect.html#1049520
[8] WebLogic Website, http://e-docs.bea.com/wls/docs81/secintro/archtect.html#1050163
[9] WebLogic Website, http://e-docs.bea.com/wls/docs81/secintro/archtect.html#1040288
[10] WebLogic Website, http://e-docs.bea.com/wls/docs81/secintro/terms.html

KAAJEE Process Flow Overview

Figure 1-1. KAAJEE & J2EE Web-based application process overview diagram



J2EE Form-based Authentication

The J2EE servlet specification provides at least two means for Web-based applications to query for end-user authentication credentials:

- Hyper Text Transport Protocol (HTTP) Basic Authentication
- J2EE Form-based Authentication

KAAJEE employs J2EE Form-based Authentication for the J2EE Web-based authentication process as part of the larger security framework. VistALink provides connectivity between KAAJEE and the VistA M Server.

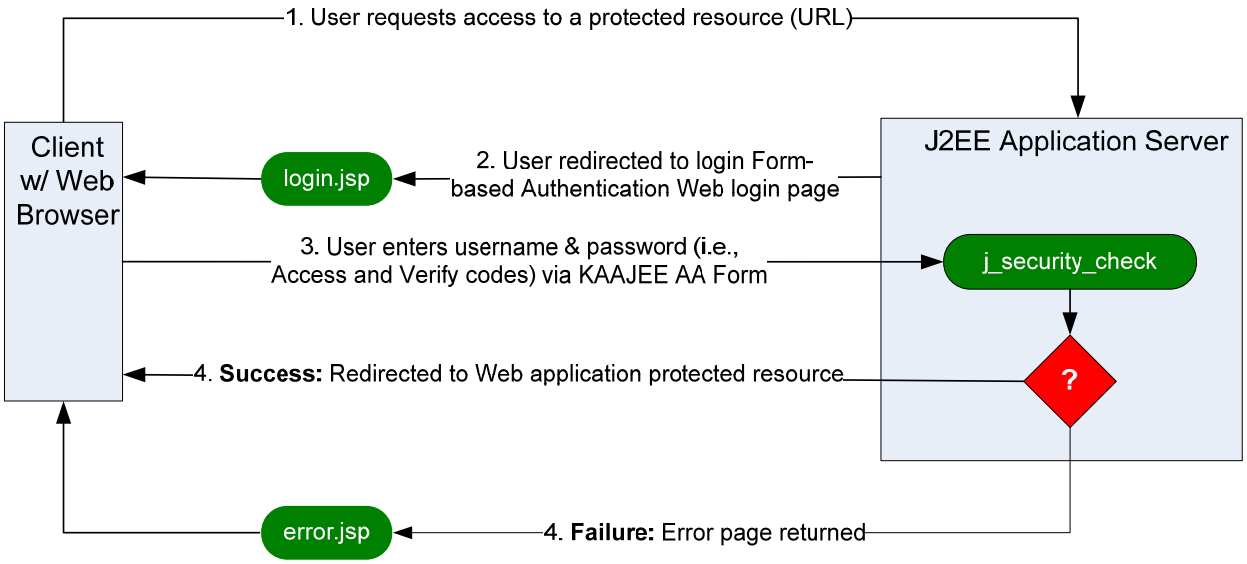
J2EE Form-based Authentication works as follows (see Figure 1-2):

1. The user on the client uses a Web browser to access a Web-based application's protected resource (URL).
2. The J2EE Application Server (container) detects that the user is not in an authenticated user session and redirects the user to the J2EE Form-based Authentication Web login page specified in the <login-config> tag in the web.xml deployment descriptor.

   NOTE: The container remembers the URL the user originally requested.

3. The user on the client submits their username and password (i.e., Access and Verify codes) via the KAAJEE Authentication and Authorization (AA) Web login form.
   a. The Web login page's responsibility is to collect user credentials (username and password) and submit them to the j_security_check "special"/"magic" servlet on the J2EE Application Server.
   b. The j_security_check "special"/"magic" servlet passes those credentials to the WebLogic Custom Security Authentication Providers.
4. J2EE Application Server authenticates the user:
   - Success:
     a. If the WebLogic Custom Security Authentication Providers authenticates the user, an authenticated session is established.
     b. If the WebLogic Custom Security Authentication Providers grants the user access to the role needed to access the originally requested page, the container redirects the user to that page.
   - Failure:
     a. If the WebLogic Custom Security Authentication Providers fails to authenticate the user, an authenticated session is *not* established.
     b. If the WebLogic Custom Security Authentication Providers does *not* grant the user access to the role needed to access the originally requested page, the container redirects the user to an error page.

Figure 1-2. Form-based Authentication overview



There *cannot* be login buttons that point directly to the login page. Only an attempt to access a protected resource (as opposed to the login page) triggers the J2EE Form-based Authentication process.

Authentication (i.e., challenging the end-user for Access and Verify codes by prompting them with the logon Web form) is triggered when an end-user attempts to access a protected Web page in the application:

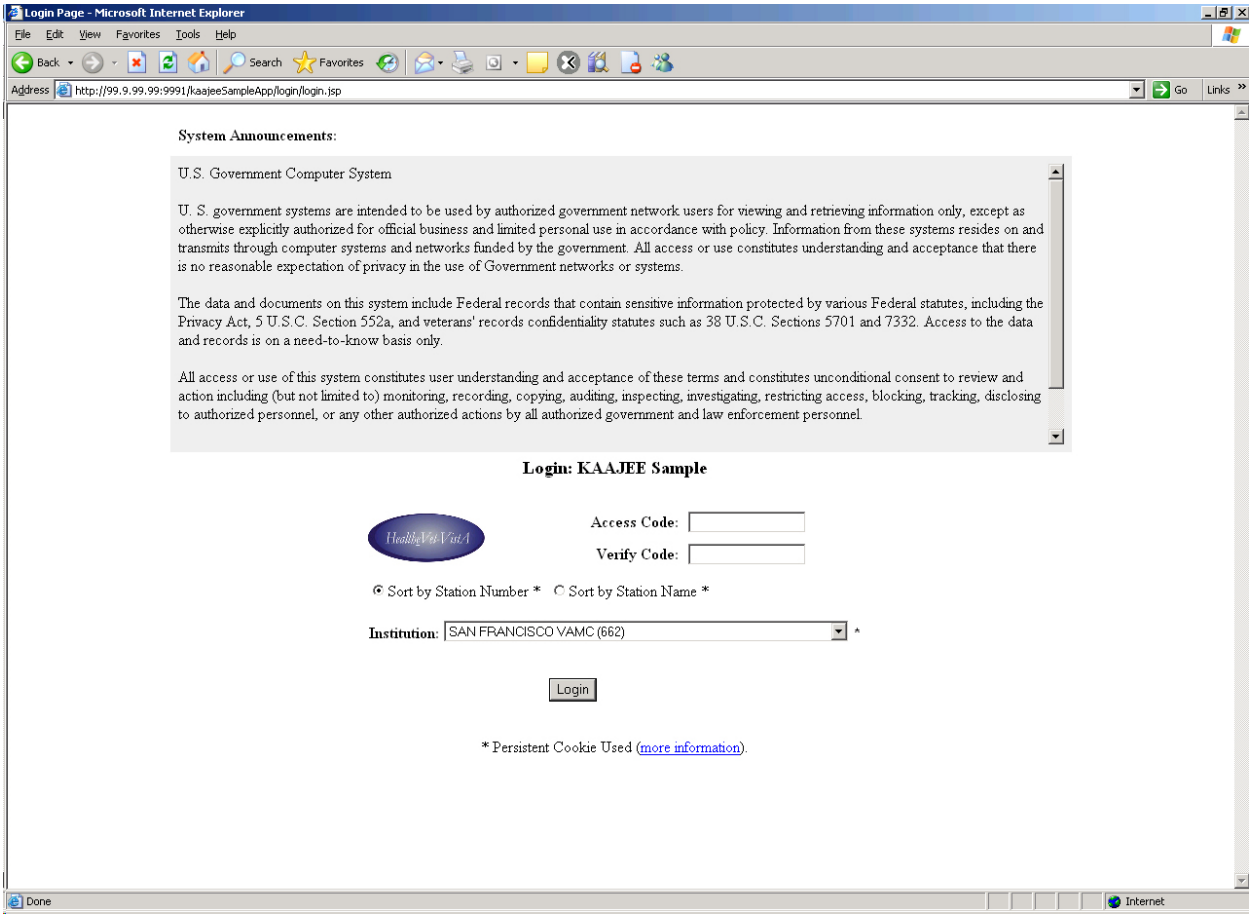
> The container will force the user to authenticate by submitting the login form only when required (for example, when an unauthenticated user tries to access a protected resource). This is termed lazy authentication, and means that users who never attempt to access a protected resource will never be forced to authenticate. Once authenticated, a user will never be challenged again within a session. The user identity will be carried through to calls to other components of the application. Therefore there is no need for user code behind protected resources to check that authentication has occurred.[11]

[11] Johnson, pg. 236.

KAAJEE J2EE Web-based Application Login Page

KAAJEE provides the official Health**e**Vet VistA J2EE Web-based application login page (i.e., login.jsp) to collect the end-user's Access and Verify codes, as well as the institution under which the user logs in. Kernel on the VistA M Server uses that information to authenticate the end-user and sign them onto VistA. A sample of the KAAJEE Web login page is displayed below:

Figure 1-3. Sample KAAJEE Web login page (i.e., login.jsp)



CAUTION: As per the Software Engineering Process Group/Software Quality Assurance (SEPG/SQA) Standard Operating Procedure (SOP) 192-039—Interface Control Registration and Approval (effective 01/29/01, see http://vista.med.va.gov/SEPG_lib/Standard%20Operating%20Procedures/192-039%20Interface%20Control%20Registration%20and%20Approval.htm), application programmers developing Health*e*Vet VistA J2EE Web-based applications that are KAAJEE-enabled *must* use the KAAJEE login Web page (i.e., login.jsp) as delivered (see Figure 1-3). Developers *must not* customize the login Web page or alter the KAAJEE software code in any way.

CAUTION: In a domain consisting of an Administration Server and several Managed Servers, the Administration Server *must* always be running, as new logins through KAAJEE will *not* succeed while the Administration Server is down.

The KAAJEE Web login page:

- Complies with Section 508 of the Rehabilitation Act Amendments of 1998.
- Provides a consistent look-and-feel across all Health**e**Vet VistA J2EE Web-based applications that are KAAJEE-enabled.

As you can see from Figure 1-3, the Introductory text (i.e., system announcement message) is displayed in the top portion of the Web login page and is preceded by the "System Announcements:" label.

Following the Introductory text, the name of the application to which you are signing on is displayed after the "Log on for:" label. Applications pass in the name of their application. In this example (Figure 1-3), the application name is **KAAJEE Sample**.

Following the application login label are the specific user entries used in order to log into the Web-based application and are described in the topic that follows (i.e., Login Procedures for J2EE Web-based Applications).

Login Procedures for J2EE Web-based Applications

To log into VistA from a J2EE Web-based application, do the following:

1. (required) Type in a valid Access code at the **Access Code** prompt.

   NOTE: Users can optionally enter both their Access and Verify codes separated by a semi-colon (**;**) at the **ACCESS Code** prompt (e.g., **accesscode;verifycode**) in order to skip entering data in the **Verify Code** prompt that follows.

2. (required if not already entered in the **Access Code** prompt, see note above) Tab to the Verify code field and type in a valid Verify code at the **Verify Code** prompt.
3. (optional) Select the sort order of the Institutions in the **Institution** dropdown list. You can sort the Institutions by Station Number or Station Name. Click on (check) either the **Sort by Station Number** or **Sort by Station Name** radio button.
4. (required) Select the appropriate Station Name/Number from the **Institution** dropdown list or accept the default value displayed.
5. (required) Click on (press) the **Login** button or press the **<Enter>** key. After the authentication process successfully completes on the VistA M Server, the requested application protected Web page will be displayed.

NOTE: The asterisks located next to the **Sort by Station Number**/**Sort by Station Name** radio buttons and the **Institution** dropdown box indicate that both the Station Name/Number sort order preference and the last Institution selected by the user are stored in a persistent cookie (see Figure 1-4). Thus, until the user changes this information, both the sort order preference and default Institution will remain the same for each subsequent login.

Login Parameter Passing for J2EE Web-based Applications

KAAJEE allows developers and end-users to pass in predefined parameters when calling a Web-based application URL. Table 1-3 defines the parameters that can be passed into the consuming application's target URL/protected page. For the examples found in the table, the application URL is represented by the **[APP_URL]** alias, which is defined as follows:

```
http://99.9.99.99:9999/kaajeeSampleApp/AppHelloWorld.jsp
```

The values indicated in the URL represent the following information:

- **99.9.99.99**—J2EE Application Server Internet Protocol (IP) address where the sample Web-based application is running.
- **9999**—J2EE Application Server Port Number where the sample Web-based application is running.
- **kaajeeSampleApp**—Application context root of the Web-based application running on the J2EE Application Server.
- **AppHelloWorld.jsp**—Name of the Web-based application protected Web page running on the J2EE Application Server.

Table 1-3. Login parameters

Parameter	Description/Usage
kaajeeDefaultInstitution	This KAAJEE login parameter is used to set the default Institution Station Number/Name on the login page. The possible values are any valid Department of Veterans Affairs Station Number (e.g., 662). The following is a sample URL passing in this parameter: **[APP_URL]**?kaajeeDefaultInstitution=662
kaajeeDisableInstitutionComponents	This KAAJEE login parameter is used to disable (grey out) both the Sort By Number/Name radio buttons and the default Institution Station Number/Name on the login page. The possible values are as follows (case sensitive): • true • false The following is a sample URL passing in this parameter: **[APP_URL]**?kaajeeDisableInstitutionComponents=true
kaajeeSortStationBy	This KAAJEE login parameter is used to set the default sort order of the Institution Station Name/Number in the Institution dropdown list on the login page. The possible values are as follows (case sensitive): • number • name The following is a sample URL passing in this parameter: **[APP_URL]**?kaajeeSortStationBy=name

Parameter	Description/Usage
kaajeeDisableSortStationBy	This parameter is used to disable (grey out) the Sort By Number/Name radio buttons on the login page. The possible values are (case sensitive): • true • false **NOTE:** If the kaajeeDisableInstitutionComponents parameter is set to "true" the Sort By Number/Name radio buttons on the login page are automatically disabled as well. The following is a sample URL passing in this parameter: `[APP_URL]?kaajeeDisableSortStationBy=true`

These parameters can be used in any combination and listed in any order, as shown in the examples that follow.

Example 1:

This example shows one parameter being passed into an application's URL to set the following:

- Default institution to "662BU"

```
http://99.9.99.99:9999/kaajeeSampleApp/AppHelloWorld.jsp?kaajeeDefaultInstitution=662BU
```

Example 2:

This example shows two parameters being passed into an application's URL to set the following:

- Default institution to "662"
- Disable the institution components

```
http://99.9.99.99:9999/kaajeeSampleApp/AppHelloWorld.jsp?kaajeeDefaultInstitution=662&kaajeeDisableInstitutionComponents=true
```

Example 3:

This example shows two parameters being passed into an application's URL to set the following:

- Default institution to "662"
- Sort Institutions by number

```
http://99.9.99.99:9999/kaajeeSampleApp/AppHelloWorld.jsp?kaajeeDefaultInstitution=662&kaajeeSortStationBy=number
```

Example 4:

This example shows two parameters being passed into an application's URL to set the following:

- Default institution to "662"
- Sort Institutions by name

```
http://99.9.99.99:9999/kaajeeSampleApp/AppHelloWorld.jsp?kaajeeDefaultInstitution=6
62&kaajeeSortStationBy=name
```

Example 5:

This example shows three parameters being passed into an application's URL to set the following:

- Default institution to "662"
- Sort Institutions by number
- Disable the sort radio buttons

```
http://99.9.99.99:9999/kaajeeSampleApp/AppHelloWorld.jsp?kaajeeDefaultInstitution=6
62&kaajeeSortStationBy=number&kaajeeDisableSortStationBy=true
```

Example 6:

This example shows three parameters being passed into an application's URL to set the following:

- Default institution to "662"
- Sort Institutions by name
- Disable the sort radio buttons

```
http://99.9.99.99:9999/kaajeeSampleApp/AppHelloWorld.jsp?kaajeeDefaultInstitution=6
62&kaajeeSortStationBy=name&kaajeeDisableSortStationBy=true
```

Example 7:

This example shows three parameters being passed into an application's URL to set the following:

- Default institution to "662"
- Sort Institutions by number
- Disable the institution components

```
http://99.9.99.99:9999/kaajeeSampleApp/AppHelloWorld.jsp?kaajeeDefaultInstitution=6
62&kaajeeSortStationBy=number&kaajeeDisableInstitutionComponents=true
```

Example 8:

This example shows three parameters being passed into an application's URL to set the following:

- Default institution to "662"
- Sort Institutions by name
- Disable the institution components

```
http://99.9.99.99:9999/kaajeeSampleApp/AppHelloWorld.jsp?kaajeeDefaultInstitution=6
62&kaajeeSortStationBy=name&kaajeeDisableInstitutionComponents=true
```

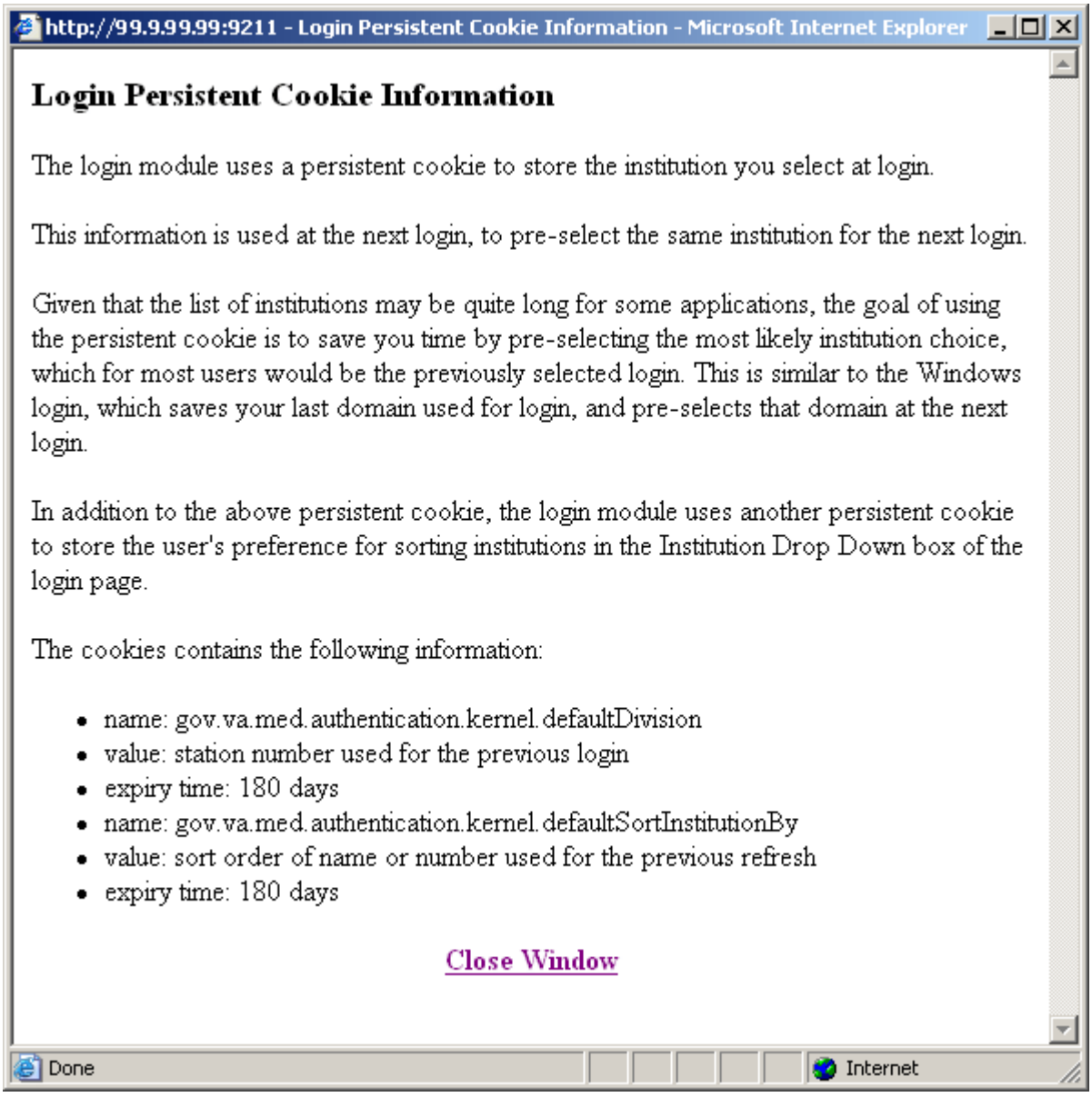
NOTE: All of these sample URLs, with various combinations of parameters, can be saved as shortcuts on your computer desktop.

Login Persistent Cookie Information

The **more information** link (i.e., "*Persistent Cookie Used [more information]"), at the bottom of the Web login page, jumps you to the "Login Persistent Cookie Information" Web page (see Figure 1-4). This Web page displays information that is stored in the persistent cookie.

For example, the persistent cookie stores your default Institution and Institution sort order preference. A sample "Login Persistent Cookie Information" Web page is shown below:

Figure 1-4. Sample login persistent cookie information



In addition to the above information, the persistent cookie also displays the Uniform Resource Locator (URL) of the application that includes the Internet Protocol (IP) address and application name (e.g., 99.9.99.99/kaajeeSampleApp).

NOTE: The KAAJEE persistent cookies are *not* stored on Terminal Servers (e.g., Citrix). The issue of using persistent cookies on Terminal Servers is that they are often not retained as part of the roaming user profile upon logout and disconnect.

As a workaround, with the added support for parameter passing of Default Institution and Institution sorting preferences users, users can create shortcuts on their desktops and use them to pass in their Default Institution and Institution sorting preferences users rather than rely on persistent cookies.

REF: For more information on parameter passing, please refer to the "Login Parameter Passing for J2EE Web-based Applications" topic in this chapter.

REF: For information on common login-related error messages, please refer to the "Common Login-related Error Messages" topic in Chapter 11, "Troubleshooting," in this manual.

For a list of other login-related error messages, please refer to the "Symptoms and Possible Solutions" topic in Chapter 7 in the *VistALink System Administration Guide*.

REF: For more information on the Kernel signon process and related error messages, please refer to the "Signon/Security" section in the *Kernel Systems Management Guide*.

2. Future Software Implementations

Outstanding Issues

The following table lists the current outstanding issues with the Kernel Authentication and Authorization Java (2) Enterprise Edition (KAAJEE) software:

Table 2-1. KAAJEE current outstanding issues

Issue	Description
Enforce Failed Login Attempt Limit	KAAJEE does not yet implement a failed login attempt limit. It's possible that modifications to the KaajeeVistaLinkConnectionSpec class could accomplish this by hooking into Kernel's IP-based failed login limit functionality.  **NOTE:** Implementing this depends on new feature that will be in a future iteration of VistALink and Kernel.

Future Enhancements

The following table lists the future enhancements for KAAJEE:

Table 2-2. KAAJEE future enhancements

Enhancement	Description
Enable CCOW Functionality	KAAJEE will be CCOW enabled for user context (under development). It will be implemented within the framework of the HL7 CCOW User Context standard. Thus, the KAAJEE architecture will allow users to authenticate and sign on to multiple applications that are CCOW-enabled and Single Sign-On/User Context (SSO/UC)-aware using a single set of credentials, which will reduce the need for multiple IDs and passwords in the Health**e**Vet clinician desktop environment. **REF:** For more information on SSO/UC, please refer to the *Single Sign-On/User Context (SSO/UC) Deployment Guide*.
Provide Helper Function for User's Default Division	The LoginUserInfoVO object could provide a helper function to retrieve a user's "default" division (as stored by the authenticating VistA M Server) in the case that the enclosing J2EE application configures KAAJEE to retrieve the <user-new-person-divisions> list at the time of authentication. **REF:** For more information on the LoginUserInfoVO object, please refer to the "LoginUserInfoVO Object" topic in Chapter 7, "Programming Guidelines," in this manual. **REF:** For more information on the <user-new-person-divisions> tag in the kaajeeConfig.xml file, please refer to the "KAAJEE Configuration File Tags" topic in Chapter 6, "KAAJEE Configuration File," in this manual.
Support Change Verify Code	KAAJEE does not currently allow users to change their Verify code when signing onto VistA via KAAJEE-enabled Web-based applications. Currently, users are presented with an error message and advised to use another VistA application to change their Verify code. **REF:** For more information on this error code, please refer to the "Error: Your verify code has expired or needs changing" topic in Chapter 11, "Troubleshooting," in this manual.
Purge KAAJEE SSPI Tables at System Startup	KAAJEE does not currently purge the SSPI tables at system startup, it only deletes and recreates individual user entries in the tables during the login process.

II. Developers Guide

This is the Developers Guide section of this supplemental documentation for Kernel Authentication and Authorization Java (2) Enterprise Edition (KAAJEE). It is intended for use in conjunction with the KAAJEE software. It details the developer-related KAAJEE documentation (e.g., developer procedures needed to incorporate the KAAJEE authorization and authentication functionality into Web-based applications, APIs exported with KAAJEE, etc.).

3. KAAJEE Installation Instructions for Developers

Preliminary Considerations: Developer Workstation Requirements

The following minimum hardware and software tools/utilities are required by developers when developing J2EE Web-based applications that are Kernel Authentication and Authorization Java (2) Enterprise Edition (KAAJEE)-enabled:

Table 3-1. Developer minimum hardware and software tools/utilities required for KAAJEE-enabled application development

Minimum Hardware/Software Requirement	Description
Workstation Hardware	80x86-based client or server workstation.
Operating System	One of the following 32-bit operating systems: • Linux (i.e., Red Hat Enterprise ES 3.0) • Microsoft Windows XP • Microsoft Windows 2000
Development-related Software	The following development-related software is required in order to develop J2EE Web-based applications that utilize KAAJEE functionality: • KAAJEE Software (see Table 1-2)—Software used to KAAJEE-enable Web-based applications. • Java 2 Standard Edition (J2SE) Java Development Kit (JDK)—COTS software for development of J2EE Web-based applications that are KAAJEE-enabled. The JDK should include Java Runtime Environment (JRE) and other developer tools to write Java code. • Health***e***Vet-VistA Web-based Software Applications (e.g., Blind Rehab, Patient Advocate Tracking System [PATS], Veterans Personal Finance System [VPFS])—Web-based software *must* be available to the end-user/developer. • Internet Browser (e.g., Microsoft Internet Explorer 6.0 or higher)—Commercial-Off-The-Shelf (COTS) software. Internet browser software *must* be available to the end-user on the client workstation. • Oracle SQL*Plus (9.2.0.1.0 or higher)—COTS software for configuring SSPI SQL on an Oracle 9i database.  **REF:** For more information on configuring files

Minimum Hardware/Software Requirement	Description
	and integrating KAAJEE with Web-based software applications, please refer to Chapter 4, "Integrating KAAJEE with an Application," in this manual.
Network Communications Software/Capability **REF:** For more information on telecommunications support, please visit the VHA Communication Services Office (CSO) Home Page: http://vaww.va.gov/cso/	All developer workstations *must* have the following network communications software and capability: • Networked client/server workstations running Microsoft's native TCP/IP stack. **NOTE:** Currently, only Winsock compliant TCP/IP protocol is supported on the LAN or remotely as Point-to-Point Protocol (PPP) or Serial Line Internet Protocol (SLIP). You *must* use RAS (Remote Access Service) or Dialup Networking to connect to the server using PPP or SLIP. For the setup of RAS or Dialup Networking, please refer to the appropriate operating system's documentation. • Connectivity with the VistA M Server (i.e., VA Wide Area Network [WAN] connectivity). Run PING.EXE to test the connectivity. • Capability to log onto the NT network using a unique NT Logon ID.

Dependencies—KAAJEE and VistALink Software

The following table shows the dependency relationships between the current version of KAAJEE, SSPIs, and VistALink software:

Table 3-2. Dependencies——KAAJEE, SSPIs, and VistALink software

Developer-related Software			Application Server Software	
Software	**Version**	**KAAJEE Software Release/Distribution**	**SSPI Software Release/Distribution**	**VistALink Software Release/Distribution**
KAAJEE	1.0.1.xxx	KAAJEE_1_0_1_XXX.ZIP	Kaajee_security_ provider_1.0.0.010.zip	VistALink 1.5 (fully patched)

REF: For a list of VistALink dependent VistA M Server patches, please refer to the *VistALink Installation Guide (Version 1.5)*.

KAAJEE Installation Instructions

The following instructions are only required for those workstations to be used by developers to develop KAAJEE-enabled Health*e*Vet-VistA Web-based software applications running on a WebLogic Application Server.

REF: For Developer Workstation platform requirements, please refer to the "Preliminary Considerations: Developer Workstation Requirements" topic in this chapter.

1. Confirm/Obtain Developer Workstation Distribution Files *(recommended)*

The following files are needed to install the KAAJEE developer-related software:

Table 3-3. Distribution files—KAAJEE developer-related software/documentation files

File Name	Type	Description
KAAJEE_1_0_1_README.TXT	ASCII	**Readme File** (manual). This file provides any pre-installation instructions, last minute changes, new instructions, and additional information to supplement the manuals. Read all sections of this file prior to following the installation instructions in the *Kernel Authentication & Authorization for J2EE (KAAJEE) Installation Guide* (i.e., KAAJEE_1_0_1_INSTALLGUIDE.PDF).
KAAJEE_1_0_1_INSTALLGUIDE.PDF	Binary	**Installation Guide** (manual). Use in conjunction with the Readme text file (i.e., KAAJEE_1_0_1_README.TXT).
KAAJEE_1_0_1_XXX.ZIP	Binary	**KAAJEE Distribution File** (jar files). This Zip file contains the KAAJEE software for development of Health*e*Vet-VistA Web-based applications requiring Authentication and Authorization against Kernel on the VistA M Server via KAAJEE.

REF: For the KAAJEE software release, all distribution files, unless otherwise noted, are available for download from the Enterprise Product Support (EPS) anonymous directories:

- Preferred Method download.vista.med.va.gov

  This method transmits the files from the first available FTP server.
- Albany OIFO ftp://ftp.fo-albany.med.va.gov/
- Hines OIFO ftp://ftp.fo-hines.med.va.gov/
- Salt Lake City OIFO ftp://ftp.fo-slc.med.va.gov/

2. Create a KAAJEE Staging Folder *(required)*

Create a KAAJEE Staging Folder on your developer workstation. This will be referred to as the **<STAGING_FOLDER>** alias for the rest of the instructions.

3. Unzip/Explode KAAJEE Software *(required)*

Unzip/Explode the KAAJEE_1_0_1_XXX.ZIP software distribution file in the **<STAGING_FOLDER>**.

After unzipping/exploding the KAAJEE_1_0_1_XXX.ZIP file, you will see the following contents/folder structure:

Table 3-4. kaajee-1.0.1.xxx—KAAJEE folder structure

Folder/Structure	Description
<root>	This folder contains the readme.txt file (manual), which includes an introduction, change history, any special installation instructions, and any known issues/limitations. **NOTE:** This file includes a description of the current KAAJEE software version numbering scheme. In the future, a separate authoritative source will be created for determining future version numbering schemes for all Health**e**Vet-VistA software file and folder names.
..\dd_examples	This folder contains the sample application deployment descriptor files (developer-related software): • application.xml **REF:** For an example of this file, please refer to Appendix A—Sample Deployment Descriptors in this manual. • kaajeeConfig.xml **REF:** For an example of this file, please refer to Chapter 6, "KAAJEE Configuration File," in this manual. • kaajeeConfig.xsd • role_mapping_worksheet.xls **REF:** For an example of this worksheet, please refer to Appendix B—Mapping WebLogic Group Names with J2EE Security Role Names in this manual. • web.xml

Folder/Structure	Description
	REF: For an example of this file, please refer to Appendix A—Sample Deployment Descriptors in this manual. • weblogic.xml **REF:** For an example of this file, please refer to Appendix A—Sample Deployment Descriptors in this manual.
..\doc	This folder contains the KAAJEE documentation (readme.txt file).
..\jars	This folder contains the KAAJEE jar files (developer-related software).
..\jars\jsp\login	This folder contains the complete set of KAAJEE Web forms for J2EE Form-based Authentication to prompt the user for their Access and Verify codes and enforce other rules related to Kernel Signon Security (e.g., Login and Login Error Web pages). These forms should be included in the application's Web root, as "/login" (developer-related software).
..\javadoc ..\javadoc\gov\va\med\authentication\kernel	This folder contains the KAAJEE API documentation (manual) for the server-side Java source code (HTML format). This folder contains the class-use subfolder that describes the KAAJEE and login classes, inner classes, interfaces, constructors, methods, fields, etc. **REF:** For more information, please review the help-doc.html file located in the ..\javadoc folder.
..\samples	This folder contains the KAAJEE Sample Web Application ear and exploded ear files. It also includes the MD5 file for software version validation purposes. In addition there is shortcuts subfile containing sample shortcut URLs.
..\src	This folder contains the KAAJEE source code (i.e., application server software).

NOTE: KAAJEE makes internal API calls to the Standard Data Services (SDS) Database/Tables 3.0 (or higher). SDS is responsible for maintaining this database and related tables.

KAAJEE 1.0.1.xxx distributes SDS 13.0 client jar files as part of the Sample Web Application. If you deploy the both the KAAJEE Sample Web Application and your own Web-based application on the same WebLogic Application Server domain instance and intend to use a different version of SDS, those client jar files will need to be swapped out for the appropriate version of the SDS client jar files. Otherwise, There may be a conflict if both applications reference the same JNDI tree.

4. Review/Use KAAJEE Files for Web-based Applications *(recommended)*

To build your Health**e**Vet-VistA J2EE Web-based applications that are KAAJEE-enabled you need to configure and include the kaajee-1.0.1.xxx.jar file located in the following directory:

<STAGING_FOLDER>\kaajee-1.0.1.xxx\jars

Each Health**e**Vet-VistA Web-based application requiring Authentication and Authorization against Kernel on the VistA M Server should use the standard KAAJEE Web login page, which is available with the login.jsp file located in the following KAAJEE directory:

<STAGING_FOLDER>\kaajee-1.0.1.xxx\jars\jsp\login\

CAUTION: Consuming applications should *not* provide a direct link to the login.jsp file. Otherwise, users could get a login error message when they click on that link.

REF: For more information on this login error message, please refer to the "Error: You navigated inappropriately to this page" topic in Chapter 11, "Troubleshooting," in this manual.

Review the sample descriptor files located in the following KAAJEE directory:

<STAGING_FOLDER>\kaajee-1.0.1.xxx\dd_examples

Use these sample descriptor files as templates for your Web-based applications.

REF: For sample descriptor files distributed with KAAJEE, please refer to Appendix A—Sample Deployment Descriptors in this manual.

REF: For more information on configuring files and integrating KAAJEE with Web-based software applications, please refer to Chapter 4, "Integrating KAAJEE with an Application," in this manual.

For example:

Figure 3-1. Sample application weblogic.xml file (e.g., KAAJEE Sample Web Application)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web
Application 7.0//EN"
"http://www.bea.com/servers/wls700/dtd/weblogic700-web-jar.dtd">
<weblogic-web-app>
<security-role-assignment>
<role-name>XUKAAJEE_SAMPLE_ROLE</role-name>
<principal-name>XUKAAJEE_SAMPLE</principal-name>
</security-role-assignment>

  <session-descriptor>
    <session-param>
      <param-name>CookieName</param-name>
      <param-value>kaajeeJSESSIONID</param-value>
    </session-param>

  </session-descriptor>

</weblogic-web-app>
```

In this sample application weblogic.xml file, the developers use KAAJEE Sample Web Application-related VistA M Server J2EE security keys and role names.

The <session-descriptor> tag contains the <session-param> tag, which defines attributes for Hyper Text Transport Protocol (HTTP) sessions, as shown in Figure 3-1.

The WebLogic Application Server defines the session cookie name. If it is not set by the user, it defaults to JSESSIONID. KAAJEE needs to set the session cookie name. You can set this to a more specific name for your application. For example:

- KAAJEE: kaajeeJSESSIONID
- ApplicationOne: applicationoneJSESSIONID
- ApplicationTwo: applicationtwoJSESSIONID

For KAAJEE to execute correctly, it needs to have a <run-as> tag, which causes it to run as an Admin user, as shown below:

Figure 3-2. Sample excerpt from a web.xml file—Using the run-as tag

```
  <servlet>
    <servlet-name>LoginController</servlet-name>
    <servlet-class>gov.va.med.authentication.kernel.LoginController</servlet-
class>
  <run-as>
  <!—In this example, weblogic is the boot user name (i.e., weblogic console
user name)  -->
<role-name>weblogic</role-name>
</run-as>
  </servlet>
```

Make sure that the application context name is in the kaajeeConfig.xml file, as shown below:

Figure 3-3. Sample <context-root-name> tag found in the kaajeeConfig.xml file

```
<context-root-name>/kaajeeSampleApp</context-root-name>
```

Congratulations! You have now completed the installation of KAAJEE-related software on the developer workstation.

4. Integrating KAAJEE with an Application

This chapter describes how application developers can modify their Health*e*Vet-VistA Web-based applications to integrate Kernel Authentication and Authorization Java (2) Enterprise Edition (KAAJEE) 1.0.1.xxx for Authentication and Authorization to the VistA M Server.

This chapter discusses the following topics:

- Assumptions When Implementing KAAJEE
- Software Requirements
- Web-based Application Procedures to Implement KAAJEE

Assumptions When Implementing KAAJEE

The following assumptions are made regarding application developers and Health*e*Vet-VistA J2EE Web-based applications when implementing KAAJEE (Iteration 1):

- **Developer Training—**It is assumed that developers have J2EE experience, including the following skills:
  - Writing Servlets
  - Configuring J2EE Deployment Descriptors
  - Deploying Java-based applications
  - Configuring WebLogic 8.1 (SP4 or higher)-specific Deployment Descriptors
  - Configuring/Using Oracle 9i database (e.g., Security Service Provider Interface [SSPI])
  - Configuring/Using Log4J
  - Implementing the security plug-in for WebLogic 8.1 (SP4 or higher) by using custom Security Service Provider Interfaces (SSPIs)

REF: Information about implementing the security plug-in and SSPIs for WebLogic 8.1 (SP4 or higher) can be found at the following references:

- *Kernel Authentication & Authorization for J2EE (KAAJEE) Installation Guide*
- WebLogic Documentation Website at the following Website:

  http://e-docs.bea.com/wles/docs42/dvspisec/index.html
- Applications using JMX to communicate to the WebLogic SSPIs Website at the following Website:

  http://edocs.bea.com/wls/docs81/jmx/basics.html#1128495

Software Requirements/Dependencies

In order to KAAJEE-enable a Web-based application, developers require the following software:

Table 4-1. Dependencies—KAAJEE software requirements for development

Category	Software	Version/Notes
Developer Workstation	Java Integrated Development Environment (IDE) Java 2 Standard Edition (J2SE) Java Development Kit (JDK)	Any version. Developer software installed on the workstation used for developing Health**e**Vet-VistA J2EE Web-based applications. The JDK should include a Java Runtime Environment (JRE) and other developer tools to write Java code.
	KAAJEE	Version 1.0.1.xxx. Developer software installed on the workstation used for developing, running, and testing Health**e**Vet-VistA KAAJEE-enabled J2EE Web-based applications (see Table 1-2).
Application Server	WebLogic	Version 8.1 (SP4 or higher).
	KAAJEE SSPIs	Version 1.0.0.010.
	VistALink	Version 1.5 (fully patched).
Database	Oracle Database	Version 9i or higher (e.g., Security Service Provider Interface [SSPI])
	SDS Tables	Version 3.0 or higher. **NOTE:** KAAJEE works with SDS 3.0 or higher; however, KAAJEE 1.0.1.xxx distributes SDS 13.0 client jar files as part of the Sample Web Application. If you deploy the both the KAAJEE Sample Web Application and your own Web-based application on the same WebLogic Application Server domain instance and intend to use a different version of SDS, those client jar files will need to be swapped out for the appropriate version of the SDS client jar files. Otherwise, There may be a conflict if both applications reference the same JNDI tree.
VistA M Server	Kernel	Version 8.0, fully patched (see Table 1-2).

NOTE: Kernel is the designated custodial software application for KAAJEE; however, KAAJEE comprises multiple patches and software releases from several Health**e**Vet-VistA applications.

REF: For the specific KAAJEE software and VistA M Server patches required for the implementation of KAAJEE, please refer to Table 1-2 in the "KAAJEE" topic in Chapter 1 in this manual.

Web-based Application Procedures to Implement KAAJEE

1. Use of VistALink to Authenticate Users Based on Configured Station Numbers

KAAJEE makes use of VistALink to authenticate a user against a specific M system, based on configured station numbers. KAAJEE relies on VistALink during the following steps:

a. Obtain the Java Naming and Directory Interface (JNDI) name of the VistALink connector pool (i.e., standard that provides a unified interface to multiple naming and directory services), based on the Station Number of the institution the user selects in the applications' Web login page. VistALink's institution mapping facility is used to return the JNDI name of the appropriate connector (and therefore destination M system) based on station number. The list of allowed authenticating Station Numbers is defined in the server-side deployment descriptor (i.e., kaajeeConfig.xml file).

b. Make RPC calls over the selected VistALink connector to the corresponding M system, to check the user's credentials (i.e., Access and Verify codes). The VistALink connector whose JNDI name was obtained in Step #1a above is used.

KAAJEE depends on institution mapping being set up for your VistALink connectors. J2EE Web-based application developers *must* set up connectors at every site they intend to support KAAJEE logins.

REF: For more information on VistALink, please consult the VistALink documentation.

2. Access VA Standard Data Services (SDS) Tables

VA Standard Data Services (SDS) has created and maintains standardized tables in a database (e.g., VA Institutions). These tables *must* be accessible to your Web-based application. The minimum version required is 3.0 or higher. KAAJEE uses the read-only Institution API and the data in the SDS Institution table to do the following:

- Retrieve institution display names.
- Retrieve child institutions.
- Verify if divisions share the same VistA M Server provider instance.

NOTE: KAAJEE works with SDS 3.0 or higher; however, KAAJEE 1.0.1.xxx distributes SDS 13.0 client jar files as part of the Sample Web Application. If you deploy the both the KAAJEE Sample Web Application and your own Web-based application on the same WebLogic Application Server domain instance and intend to use a different version of SDS, those client jar files will need to be swapped out for the appropriate version of the SDS client jar files. Otherwise, There may be a conflict if both applications reference the same JNDI tree.

Therefore, the following are required:

- A Connection Pool and a Data Source needs to be created on the application server to point to the database housing the SDS tables.

  To configure the SDS tables for a J2EE DataSource, please refer to the "Configuring for a J2EE DataSource" topic in the *SDS API Installation Guide*.

  REF: The *SDS API Installation Guide* is included in the SDS software distribution ZIP files, which are available for download at the following Website:

  http://vaww.sts.infoshare.va.gov/STS_SDS/Project%20Artifacts/Forms/AllItems.aspx

- The jdbc.properties file needed by the SDS read-only API *must* be in your application's classpath at the location expected by the API.

  KAAJEE distributes two sample versions of the jdbc.properties file, depending on the operating system. These sample files are located in the following distribution directory:

  <STAGING_FOLDER>/kaajee-1.0.1.xxx/samples/exploded/kaajeeSampleApp-1.0.1.xxxEAR/APP-INF/classes/gov/va/stddata/factory/db

Figure 4-1. Sample jdbc.properties.cache file

```
jdbc.url=jdbc:Cache://127.0.0.1:1972/SDS
jdbc.driver=com.intersys.jdbc.CacheDriver
user=_SYSTEM
password=SYS
```

Figure 4-2. Sample jdbc.properties.oracle file

```
jdbc.url=jdbc:oracle:thin:@<DB-HOST>:1521:<DB-NAME>
jdbc.driver=oracle.jdbc.driver.OracleDriver
user=<SDSUSER_ID>
password=<SDSUSER_PASSWORD>
```

NOTE: Depending on the operating system, you can use either of these sample files; however, make sure you substitute the values appropriate to your system and rename the file to **jdbc.properties**.

- The SDS read-only API 3.0 (or higher) *must* itself be available in your application's classpath. KAAJEE 1.0.1.xxx distributes the following two SDS 13.0 client jar files as part of the Sample Web Application:
  - vha-stddata-client-13.0.jar
  - vha-stddata-basic-13.0.jar

NOTE: KAAJEE works with SDS 3.0 or higher; however, KAAJEE 1.0.1.xxx distributes SDS 13.0 client jar files as part of the Sample Web Application. If you deploy the both the KAAJEE Sample Web Application and your own Web-based application on the same WebLogic Application Server domain instance and intend to use a different version of SDS, those client jar files will need to be swapped out for the appropriate version of the SDS client jar files. Otherwise, There may be a conflict if both applications reference the same JNDI tree.

REF: For more information on the use of the SDS APIs, please refer to the *SDS API Installation Guide*. The SDS documentation is included in the SDS software distribution ZIP files, which are available for download at the following Website:

http://vaww.sts.infoshare.va.gov/STS_SDS/Project%20Artifacts/Forms/AllItems.aspx

3. Import KAAJEE Jar File

The following jar file is present in the STAGING_FOLDER>\kaajee-1.0.1.xxx\jars folder of the KAAJEE distribution zip file (i.e., KAAJEE_1_0_1_XXX.ZIP):

Table 4-2. KAAJEE jar distribution file

Jar File Name	Description
kaajee-1.0.1.xxx.jar	The KAAJEE java classes.

To import this library into your development environment, add this jar to the compiler paths of your Integrated Development Environment (IDE), ANT configuration, and/or anywhere else in your development environment that needs to know classpaths.

Table 4-3. Jar files and classpath defined for KAAJEE-enabled Web-based applications

Classpath	Description
kaajee-1.0.1.xxx.jar	KAAJEE developer-related software.
j2ee.jar	J2EE java classes.
jaxen-full.jar	XML software.
log4j-1.2.8.jar	Log file software.
saxpath.jar	XML SAX parser.

Classpath	Description
weblogic.jar	WebLogic API.

The kaajee-1.0.1.xxx.jar file *must* be distributed in your application's Enterprise Archive (.ear) file with an application-level classloader.

When you are ready to deploy/distribute your application, perform the following steps:

a. (required) Package the kaajee-1.0.1.xxx.jar file (see Table 4-2) in your application's ear file (e.g., in a "../APP-INF/lib" folder descendent from the root level of your application's ear file).

b. (required) Ensure that kaajee-1.0.1.xxx.jar is *not* located in a deeper level of the classloader hierarchy than that of an application, *anywhere* on the application server. Otherwise, the singletons will be instantiated with settings inappropriate for your application, and the KAAJEE security system will function inappropriately for your application.

4. Import Other Dependent Jar Files

KAAJEE-enabled Web-based applications also have dependencies on the following jar files:

Table 4-4. Other dependent jar files for KAAJEE-enabled Web-based applications

Jar File Name	Description
Log4J.jar	(optional) A logging utility from the Apache Jakarta Project. **NOTE:** The Jakarta Project creates and maintains open source solutions on the Java platform for distribution to the public at no charge. **REF:** For more information on the Jakarta Project, please visit the following Website: http://jakarta.apache.org/
vha-stddata-basic-13.0.jar	(required) Two Standard Data Services (SDS) jar files (as of Version 13.0).
vha-stddata-client-13.0.jar	
vljConnector-1.5.1.**xxx**.jar	(required) VistALink RAR connector. **NOTE:** Substitute the current VistALink build version number for the "**xxx**" in the file name.

Jar File Name	Description
vljFoundationsLib-1.5.1.**xxx**.jar	(required) VistALink library. **NOTE:** Substitute the current VistALink build version number for the "**xxx**" in the file name.

To import these libraries into your development environment, add all jars to the compiler paths of your IDE, ANT configuration, and/or anywhere else in your development environment that needs to know classpaths.

Once you install VistALink on a WebLogic Application Server, both VistALink and Log4J libraries are available on a classloader that is parent to all other applications; therefore, you do not need to export these jar files in your application.

You do, however, need to export the SDS jar files. Because they are used by the kaajee-1.0.1.xxx.jar, they need to be loaded via an application-level classloader in order for the kaajee-1.0.1.xxx.jar to have visibility to them.

Thus, when you deploy/distribute your application it is recommended that you distribute both SDS jar files in the same ear file location as you distribute the kaajee-1.0.1.xxx.jar file.

5. Import KAAJEE Login Folder

The following files are present in the "login\" folder contained in the **<STAGING_FOLDER>**\kaajee-1.0.1.xxx\jars\jsp folder of the KAAJEE distribution zip file (i.e., KAAJEE_1_0_1_XXX.ZIP):

Table 4-5. KAAJEE login folder files

Directory	File Name	Description
..login\	login.jsp	Login Web page for authentication. This is the Login Web page where users enter their Access and Verify codes and choose an Institution from a dropdown list. **CAUTION: Consuming applications should *not* provide a direct link to the login.jsp file. Otherwise, users could get a login error message when they click on that link, see the description for navigatonerrordisplay.jsp in this table.**
..login\	loginCookieInfo.htm	Login persistent cookie information.
..login\	loginerror.jsp	J2EE Form-based Authentication error display Web page for failure to authenticate J2EE Application Server login credentials.
..login\	loginerrordisplay.jsp	Login error display Web page for failure to authenticate VistA M Server login credentials. **REF:** For more information on these types of errors, please refer to Chapter 11, "Troubleshooting," in this manual.
..login\	navigatonerrordisplay.jsp	Error display Web page displayed after a user successfully logs into a Web application and then presses the browser **Back** button to get back to the KAAJEE Web login page. **REF:** For more information on this error, please refer to the "Error: You navigated inappropriately to this page" topic in Chapter 11, "Troubleshooting," in this manual.

Directory	File Name	Description
..login\	SessionTimeout.jsp	Login session timeout Web page.
..login\images\	HealtheVetVistaSmallBlue.jpg	Health**e**Vet-VistA small blue logo image file.
..login\images\	HealtheVetVistaSmallWhite.jpg	Health**e**Vet-VistA small white logo image file.
..login\javascript\	login.js	This JavaScript file supports functions associated with the login.jsp file. For example: • Sorting of Institutions. • Enabling/Disabling of components as part of login parameter passing.

Import the entire "login\" folder, including the folder itself, into your Web-based application. These files *must* be brought into your J2EE Web-based application, and distributed with it, because by the J2EE standard, any pages that are used in J2EE Form-based Authentication *must* run in the same context as the Web-based application:

REF: For more information on how to configure your web.xml file for the login folder, please refer to "5. Configure Web-based Application for J2EE Form-based Authentication" topic in Chapter 5, "Role Design/Setup/Administration," in this manual.

6. Set Up KAAJEE Configuration File

KAAJEE relies on a configuration file (i.e., kaajeeConfig.xml file) to read in all administrator-configurable settings.

You can use the kaajeeConfig.xml file that is distributed with the KAAJEE software or you can create a KAAJEE configuration file in your J2EE Web-based application and export it along with your Web-based application.

REF: For a sample kaajeeConfig.xml file, please refer to Figure 6-2 in Chapter 6, "KAAJEE Configuration File," in this manual.

If you create a new KAAJEE configuration file, do the following:

a. (required) Create an empty XML file within your Web-based application's context root (e.g., in the WEB-INF folder). The developer can choose any name for this XML file.

b. (required) Set the top-level tag for the file to <kaajee-config>. For example:

Figure 4-3. Sample empty KAAJEE configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<kaajee-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="kaajeeConfig.xsd">

</kaajee-config>
```

c. (required) Configure the file created in the previous step (i.e., Step #6b) by following guidelines in Chapter 6, "KAAJEE Configuration File," in this manual. At a minimum, the following tags *must* be configured (see Table 6-1):

- <kaajee-config>.
- <login-station-numbers> (controls the login Web page's Institution dropdown list).
- <context-root-name>.

NOTE: For every login Station Number you enter here, you also need to use VistALink's Institution Mapping to associate that login Station Number with a VistALink connector.

REF: For more details, please refer to Chapter 6, "KAAJEE Configuration File," in this manual.

7. Configure KAAJEE Initialization Servlet (web.xml file)

You can place the KAAJEE configuration file anywhere within your Web-based application's context root. KAAJEE provides an initialization servlet to initialize KAAJEE.

The classname of the servlet is:

gov.va.med.authentication.kernel.InitKaajeeServlet

This servlet in the web.xml file is used to:

- Pass the location and name of the KAAJEE configuration file (see Figure 4-4) as a servlet parameter named:

  kaajee-config-file-location

- Control the sequence of startup using the <load-on-startup> tag.

For example:

Figure 4-4. Sample excerpt of the KAAJEE web.xml file—Initialization servlet

```
  <servlet>
   <servlet-name>KaajeeInit</servlet-name>
   <servlet-class>gov.va.med.authentication.kernel.InitKaajeeServlet</servlet-class>
   <init-param>
     <param-name>kaajee-config-file-location</param-name>
     <param-value>/WEB-INF/kaajeeConfig.xml</param-value>
   </init-param>
   <load-on-startup>3</load-on-startup>
  </servlet>
```

REF: For a sample web.xml file, please refer to "Appendix A—Sample Deployment Descriptors" in this manual.

8. Configure KAAJEE LoginController Servlet (web.xml file)

The kaajee-1.0.1.xxx.jar file includes one servlet that you *must* configure in your J2EE Web-based application's web.xml file. This servlet is referenced by the Web forms in the \login folder.

The servlet *must* be mapped to the url-pattern "/LoginController".

Configure the servlet in your application's web.xml file, as shown below:

Figure 4-5. Sample excerpt of the KAAJEE web.xml file—LoginController servlet configuration

```
  <servlet>
    <servlet-name>LoginController</servlet-name>
    <servlet-class>gov.va.med.authentication.kernel.LoginController</servlet-class>
  <run-as>
  <!—In this example, weblogic is the boot user name (i.e., weblogic console user
name)  -->
<role-name>weblogic</role-name>
</run-as>
  </servlet>

  <servlet-mapping>
    <servlet-name>LoginController</servlet-name>
    <url-pattern>/LoginController</url-pattern>
  </servlet-mapping>
```

9. Configure KAAJEE Listeners (web.xml file)

KAAJEE has two similar listeners, both of which perform logout actions for a user. Both of these listeners are available in case one listener does not work with a specific container/platform (e.g., WebLogic):

Table 4-6. KAAJEE listeners

Listener	Description
KaajeeSessionAttributeListener	The KaajeeSessionAttributeListener listens for specific (individual) session attributes that are targeted for removal, which signals a user session ending, and performs user logout actions.
KaajeeHttpSessionListener	The KaajeeHttpSessionListener listens for session destruction. It is looking for the whole session being destroyed and performs user logout actions.

KAAJE uses two different approaches to configure the listeners for future compatibility. While an HttpSessionAttributeListener method would be expected to be the way to retrieve the value of an attribute (in the case of the LoginUserInfoVO object) as a user session is destroyed[12], the HttpSessionListener's sessionDestroyed method is used to provide this functionality.

Configure these listeners in your application's web.xml file as follows (listeners in bold typeface):

Figure 4-6. Sample excerpt of the KAAJEE web.xml file—Listener configuration

```
<listener>
  <listener-class>
    gov.va.med.authentication.kernel.KaajeeSessionAttributeListener
  </listener-class>
</listener>

<listener>
  <listener-class>
    gov.va.med.authentication.kernel.KaajeeHttpSessionListener
  </listener-class>
</listener>
```

10. Design/Set Up Application Roles

Some preparation is required to correctly set up application roles. The following areas are involved:

[12] Hall, Marty, *More Servlets and Java Server Pages*, 2002, pg. 523.

- WebLogic group mappings (weblogic.xml).

  REF: For a sample spreadsheet showing a mapping between WebLogic group names (i.e., principals) with J2EE security role names, please refer to "Appendix B—Mapping WebLogic Group Names with J2EE Security Role Names" in this manual.

- VistA M Server J2EE security keys (correspond to WebLogic server group names).
- J2EE security role declarations (web.xml and weblogic.xml).
- Security constraints using J2EE security role and group names (weblogic.xml).

REF: For more detailed role configuration instructions, please refer to Chapter 5, "Role Design/Setup/Administration," in this manual.

11. Configure Log4J Logging for KAAJEE

KAAJEE uses Log4J to log error and debugging information. It is strongly recommended that you configure your application to use Log4J (in addition to any other logging system your application is using) in order to gain access to the error and debugging information produced by KAAJEE.

Configure Log4J logging so that KAAJEE error and/or debug messages are logged to the same file used by *all* J2EE-based applications running in the same domain on the application server. This assists users on the application server to monitor and troubleshoot KAAJEE and all other J2EE-based applications in one place.

REF: For specific directions on setting up logging for KAAJEE, please refer to the "Log4J Configuration" topic in Chapter 8, "Implementation and Maintenance (J2EE Site)," in this manual.

12. Protect KAAJEE Web Pages

At this point, your application is configured with KAAJEE, but has *not* yet been configured to protect any Web pages using KAAJEE. To authenticate and authorize users with KAAJEE, you need to protect the Web pages in your application by configuring J2EE Form-based Authentication in your application's web.xml file.

Once you protect your application Web pages, KAAJEE is activated. When a user tries to access a protected Web page, if all is configured correctly, the user is redirected to the KAAJEE Web login page for Authentication and Authorization.

REF: For information on setting up KAAJEE to protect Web pages, please refer to Chapter 5, "Role Design/Setup/Administration," in this manual.

5. Role Design/Setup/Administration

Protected resources in the various development environments are as follows:

- M—Menus act as protected resources and VistA M Server J2EE security keys act as groups
- Web-based applications (Kernel Authentication and Authorization Java (2) Enterprise Edition [KAAJEE])—Static Web pages, servlets, jsps, etc.
- Rich client-based applications (Fat Client Kernel Authentication and Authorization [FatKAAT])—Stateless session Enterprise JavaBeans (EJBs)

REF: This document will only discuss Web-based applications. For rich client-based applications, please refer to the FatKAAT-related software and documentation.

Roles can be assigned to the protected resources. The web.xml file lists all of these roles in addition to listing the Web protected resources and their associated roles. The web.xml file is used declaratively to filter access to protected resources based on authorized roles. Further detailed authorization can be done programmatically with the isUserInRole(role_name) method.

The BEA weblogic.xml file maps roles to principals (i.e., user and/or groups); however, KAAJEE only uses groups. Principals are physical in that they pertain to physical users. The role acts as a lock on a protected resource and the key is the principal. Only certain principals can open a lock (i.e., only those principals that are mapped to the role/lock). Since KAAJEE only uses groups and groups equate to VistA M Server J2EE security keys, then a user in M can have several security keys and some, if any, may open the role/locks in the J2EE world.

Some setup is required to correctly set up application roles. The following steps are involved:

1. Declare Groups (weblogic.xml file)
2. Create VistA M Server J2EE security keys Corresponding to WebLogic Group Names
3. Declare J2EE Security Role Names
4. Map J2EE Security Role Names to WebLogic Group Names (weblogic.xml file)
5. Configure Web-based Application for J2EE Form-based Authentication
6. Protect Resources in Your J2EE Application
7. Grant Special Group to All Authenticated Users (Magic Role)
8. Administer Users
9. Administer Roles

REF: For a sample spreadsheet showing a mapping between WebLogic group names (i.e., principals) with J2EE security role names, please refer to "Appendix B—Mapping WebLogic Group Names with J2EE Security Role Names" in this manual.

REF: For samples of the web.xml and weblogic.xml files, please refer to "Appendix A—Sample Deployment Descriptors" in this manual.

KAAJEE includes a "magic" role (i.e., AUTHENTICATED_KAAJEE_USER).

REF: For more information on the "magic" role, please refer to "7. Grant Special Group to All Authenticated Users (Magic Role)" in this chapter.

1. Declare Groups (weblogic.xml file)

KAAJEE roles are based on the group names in your application's weblogic.xml file.

For example:

Figure 5-1. Sample application weblogic.xml file with group information (e.g., KAAJEE Sample Web Application)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web
Application 7.0//EN"
"http://www.bea.com/servers/wls700/dtd/weblogic700-web-jar.dtd">
<weblogic-web-app>
<security-role-assignment>
<role-name>XUKAAJEE_SAMPLE_ROLE</role-name>
<principal-name>XUKAAJEE_SAMPLE</principal-name>
</security-role-assignment>

  <session-descriptor>
    <session-param>
      <param-name>CookieName</param-name>
      <param-value>kaajeeJSESSIONID</param-value>
    </session-param>

  </session-descriptor>

</weblogic-web-app>
```

The <principal-name> tag is the group name and also the VistA M Server J2EE Security Key name (see 2. Create VistA M Server J2EE security keys Corresponding to WebLogic Group Names). In this example, the group name is "**XUKAAJEE_SAMPLE**" and the role name is "**XUKAAJEE_SAMPLE_ROLE**."

Developers *must* place the weblogic.xml file in the application's **<WEBROOT>**\WEB-INF folder, if not already present.

NOTE: The **<WEBROOT>** represents the root directory of the application war file, if exploded.

Developers should distribute the weblogic.xml file in the WEB-INF folder in the application's war file; this war file is in the ear file.

2. Create VistA M Server J2EE security keys Corresponding to WebLogic Group Names

At user login, KAAJEE uses the XUS ALLKEYS RPC (added with Kernel Patch XU*8.0*337) to get all VistA M Server J2EE security keys associated with the user.

KAAJEE returns all VistA M Server J2EE security keys. KAAJEE then caches the results in the Oracle database and uses those security keys along with the security roles in the application's weblogic.xml file as the basis for subsequent authorization decisions.

Therefore, for every WebLogic group name in the weblogic.xml file, if a user is to be authorized to the J2EE security role that maps to the WebLogic group name (see #3. Declare J2EE Security Role Names below), the user *must* be granted a VistA M Server J2EE Security Key whose name corresponds precisely to the WebLogic group name found in the weblogic.xml file. Application developers *must* also make sure that they set the SEND TO J2EE field (#.05) in the SECURITY KEY file (#19.1) to YES for those corresponding VistA M Server J2EE security keys.

NOTE: To set the SEND TO J2EE field (#.05), use VA FileMan's Enter or Edit File Entries option [DIEDIT].

Regardless of whether a particular user is assigned a particular security key, the entire set of application-specific VistA M Server J2EE security keys corresponding to the entire set of weblogic.xml group names should be exported by your application to all VistA M Servers that would be used for authentication for your application.

3. Declare J2EE Security Role Names

In the simplest implementation, J2EE role names used by your application have exactly the same name as the corresponding WebLogic group names found in your application's weblogic.xml file (see Figure 5-1). In such cases, no mapping is required to link J2EE security role names to WebLogic group names.

4. Map J2EE Security Role Names to WebLogic Group Names (weblogic.xml file)

The security role is mapped to the group, where the group is a collection of users. This mapping is done in the weblogic.xml file (Figure 5-1); however, as long as the <role-name> tags of a security role match one-to-one with names in the <principal-name> tag in the weblogic.xml file, no mapping is needed.

REF: For a sample spreadsheet showing a mapping between WebLogic group names (i.e., principals) with J2EE security role names, please refer to "Appendix B—Mapping WebLogic Group Names with J2EE Security Role Names" in this manual.

5. Configure Web-based Application for J2EE Form-based Authentication

J2EE Form-based Authentication *cannot* be directly invoked. Instead, it is triggered by a user's attempted access to a protected page. Thus, if you need the user's identity, then all Web pages that need that identity should be protected by a security constraint in order to trigger the J2EE Form-based Authentication login process.

To configure J2EE Form-based Authentication for the application's protected resource, use the <auth-method> begin and end tags with a value of "FORM." Also, configure the location of the form-login-page and form-error-page, as shown below:

Figure 5-2. Sample excerpt of the KAAJEE web.xml file—J2EE Form-based Authentication configuration setup

```
<login-config>
<auth-method>FORM</auth-method>
 <form-login-config>
   <form-login-page>/login/login.jsp</form-login-page>
   <form-error-page>login/loginerror.jsp</form-error-page>
 </form-login-config>
</login-config>
```

NOTE: Because of the way J2EE Form-based Authentication works, there cannot be login buttons that point directly to the Web login page. Only an attempt to access a protected resource—as opposed to the Web login page, which cannot be protected since it *must* be accessed prior to successful authentication—triggers the J2EE Form-based Authentication process.

6. Protect Resources in Your J2EE Application

Resource methods (e.g., Web URLs) can now be protected using both declarative security (i.e., the standard J2EE deployment descriptor settings) and programmatic security.

For example, for Web pages, add the following to protect a particular URL:

Figure 5-3. Sample web.xml file excerpt—Protecting an application URL (e.g., KAAJEE Sample Web Application)

```
<security-constraint>
 <web-resource-collection>
   <web-resource-name>A Protected Page</web-resource-name>
   <url-pattern>/AppHelloWorld.jsp</url-pattern>
   <http-method>GET</http-method>
   <http-method>POST</http-method>
   </web-resource-collection>
   <auth-constraint>
     <role-name>XUKAAJEE_SAMPLE_ROLE</role-name>
   </auth-constraint>
   <user-data-constraint>
     <transport-guarantee>NONE</transport-guarantee>
   </user-data-constraint>
</security-constraint>
```

Once a user tries to access a protected Web page resource, for example, the login process is triggered.

7. Grant Special Group to All Authenticated Users (Magic Role)

A new group with the following name is automatically granted to all KAAJEE-authenticated users:

AUTHENTICATED_KAAJEE_USER

This "magic" role can be used to secure pages for users that do *not* otherwise have any special VistA M Server J2EE security keys granted but that need to access your application. This allows you to identify such users by still triggering the authentication process via a role security constraint.

NOTE: In order to use this magic role in an application, KAAJEE software declared this group name in the KaajeeManageableLoginModuleImpl.java file in the KAAJEE SSPI software. It is also made available as a J2EE security role in the standard J2EE deployment descriptor(s) as well.

8. Administer Users

Users simply need to be active, enabled users on a VistA M Server (one that is also configured to be one of the systems against which logins can be performed).

The existing Kernel user management tools are used to manage the divisions that are permissible for users to log into at any given site.

All users on each VistA M Server who are going to log in through KAAJEE *must* have the XUS KAAJEE WEB LOGON "B"-type option. Kernel Patch XU*8.0*329 exported and linked this option with the XUCOMMAND menu. Since all authenticated users have access to XUCOMMAND, this linkage enables all users to have access to all RPCs listed under the XUS KAAJEE WEB LOGON "B"-type option.

9. Administer Roles

J2EE roles are administered as VistA M Server J2EE security keys on the VistA M Server on which a given user has an account. To assign a J2EE role to the user, simply create (if needed) a VistA M Server J2EE Security Key with the same name as the J2EE principal (WebLogic group) that you wish to grant, and then grant the VistA M Server J2EE Security Key to the end-user.

VistA M Server security keys are non-hierarchical; hence, the roles implemented via VistA M Server J2EE security keys are also non-hierarchical. This matches J2EE security roles themselves, which are also flat.

NOTE: VistA M Server security keys are *not* multi-divisional; therefore, KAAJEE roles based on VistA M Server J2EE security keys are also *not* multi-divisional. Because of the use of the VistA M Server J2EE Security Key mechanism, for whatever divisions a user has rights to log into at one division, the end-user will have the same roles at any other division of an integrated site that the end-user is given permission by the IRM system manager to log into.

6. KAAJEE Configuration File

The kaajeeConfig.xml file controls a number of settings necessary for Kernel Authentication and Authorization Java (2) Enterprise Edition (KAAJEE) to operate. It is located in the following directory:

<STAGING_FOLDER>\kaajee-1.0.1.xxx\dd_examples

The tag sequence within the kaajeeConfig.xml file is not significant; however, this file *must* parse as a valid XML file.

KAAJEE Configuration File Tags

The kaajeeConfig.xml file has the following tags and default values:

Table 6-1. KAAJEE configuration file (i.e., kaajeeConfig.xml) tag settings

Tag Name	Description

<kaajee-config>	Root XML tag. For example:
	`<kaajee-config`
	`xmlns:xsi=`
	`"http://www.w3.org/2001/XMLSchema-instance"`
	`xsi:noNamespaceSchemaLocation=`
	`"kaajeeConfig.xsd">`
	`</kaajee-config>`

<host-application-name>	The login Web page uses this value to prominently display your application name, so that users know why they are seeing the login Web page. For example: `<host-application-name>`**`KAAJEE Sample`** `</host-application-name>`

Tag Name	Description
<login-station-numbers>	This tag contains the sub-tags (i.e., <station-number> tags) that are used to store a set of Station Numbers to present to a user at login time. It is administrator configurable.

<station-number>
(repeated **n** times)
	Within the <login-station-numbers> tag, add one <station-number> tag for every Station Number that is valid for the user to log into, for your application. You can specify both division-level and facility-level Station Numbers, as appropriate for your application. The values entered *must* be valid and recognized by Standard Data Services (SDS).
	
	NOTE: When a user selects a division to log into, KAAJEE uses this as the Station Number parameter it passes to VistALink's Institution Mapping to retrieve a JNDI connector name for VistALink; therefore, every login station number should have a mapping configured in VistALink's Institution Mapping.
	
	As distributed:
	`<login-station-numbers>`
	`<station-number>###</station-number>`
	`<station-number>###**9XX**</station-number>`
	`<station-number>###**9XX**</station-number>`
	`<station-number>###**XX**</station-number>`
	`<station-number>###**XX**</station-number>`
	`<station-number>###</station-number>`
	`<station-number>###**9XX**</station-number>`
	`<station-number>###**9XX**</station-number>`
	`<station-number>###**XX**</station-number>`
	`<station-number>###**XX**</station-number>`
	`</login-station-numbers>`
	
	Sample entries:
	`<login-station-numbers>`
	`<station-number>**11000**</station-number>`
	`<station-number>**459**</station-number>`
	`<station-number>**523**</station-number>`
	`<station-number>**631**</station-number>`
	`<station-number>**662BU**</station-number>`
	`</login-station-numbers>`
	
	NOTE: In this example, 11000 is not a valid station number that is recognized by SDS and would not be available for selection by the user at signon.
	
	NOTE: For more information on editing the login Station Numbers in the kaajeeConfig.xml file, please refer to the "Edit the KAAJEE Configuration File" topic in the *KAAJEE Installation Guide*.

Tag Name	Description
<context-root-name>	This tag is used to generate the stored username in WebLogic's LDAP directory, not as the actual context root name for the application. The <context-root-name> must be "**/**" followed by at least four characters. For example: <context-root-name>**/kaajeeSampleApp**</context-root-name> The KAAJEE code explicitly takes the 2nd through 5th characters to use as the username prefix.

<system-announcement>	This tag is an administrator-configurable logon banner. It is the introductory text displayed to users when they sign onto the system.
	KAAJEE was developed for centralized (national) applications/systems, where the main database (not M-based) and the application server are co-located; therefore, there is a one-to-many relationship between the application server and VistA M Servers. Because the presentation of the introductory text comes before the user signs into any VistA M Server and selects the Institution/Division, this text cannot be derived from a specific VistA M Server but *must* come from the application server. Thus, this tag is an administrator-configurable logon banner. It holds the introductory text displayed to users when they sign onto the system via one of these centralized KAAJEE-enabled applications.
	Sites *must* enter announcement text in this tag. Use a tilde (**~**) character to provide line breaks, or "**~ ~**" (each tilde separated by a space) to provide a paragraph break.
	For example:
	`<system-announcement>`
	`  My System Announcement~`
	`  Line 2~ ~`
	`  Paragraph 2`
	`</system-announcement>`
	  **REF:** For another example of introductory text, please refer to the "Suggested System Announcement Text" topic in this chapter.

<user-new-person-divisions>	Some applications want to support division switching only to those divisions that an IRM system manager has configured as valid divisions in a person's NEW PERSON file (#200) entry on their host VistA M Server. Defaults to "false" (case sensitive). To tell KAAJEE to return this list of divisions after login in the LoginUserInfoVO object, set the retrieve attribute of this tag to "true" (case sensitive): `<user-new-person-divisions retrieve="true" />`

Tag Name	Description
<computing-facility-divisions>	Some applications want to support division switching for all divisions supported at the same computing facility as the login division, regardless of whether explicit access has been granted to the user for any particular division. Defaults to "false" (case sensitive). To tell KAAJEE to return this list of divisions in the LoginUserInfoVO object, set the retrieve attribute tag of this tag to "true" (case sensitive): `<computing-facility-divisions retrieve="true" />`
<cactus-insecure-mode>	Enables an application with valid Access/Verify code login credentials to retrieve a non-expiring "temporary" j_username/j_password credential to use for unit testing (e.g., testing with the CACTUS unit testing framework). Defaults to "**false**" (case sensitive). For example: `<cactus-insecure-mode enabled="false" />`  **As the tag name indicates, setting this mode decreases system security. This mode should *never* be enabled on a production system. It defaults to "false" unless enabled is specifically set to "true" (case sensitive).**  **REF:** For more information on CACTUS testing, please refer to Chapter 10, "Cactus Testing with KAAJEE," in this manual.

Suggested System Announcement Text

The following is suggested text for a mandatory banner warning from the Office of Cyber and Information Security (OCIS) as of February 20, 2002:[13]

Figure 6-1. Mandatory OCIS banner warning message

U.S. Government Computer System

U. S. government systems are intended to be used by authorized government network users for viewing and retrieving information only, except as otherwise explicitly authorized for official business and limited personal use in accordance with policy. Information from these systems resides on and transmits through computer systems and networks funded by the government. All access or use constitutes understanding and acceptance that there is no reasonable expectation of privacy in the use of Government networks or systems.

The data and documents on this system include Federal records that contain sensitive information protected by various Federal statutes, including the Privacy Act, 5 U.S.C. Section 552a, and veterans' records confidentiality statutes such as 38 U.S.C. Sections 5701 and 7332. Access to the data and records is on a need-to-know basis only.

All access or use of this system constitutes user understanding and acceptance of these terms and constitutes unconditional consent to review and action including (but not limited to) monitoring, recording, copying, auditing, inspecting, investigating, restricting access, blocking, tracking, disclosing to authorized personnel, or any other authorized actions by all authorized government and law enforcement personnel.

Unauthorized user attempts or acts to (1) access, upload, change, or delete information on this system, (2) modify this system, (3) deny access to this system, (4) accrue resources for unauthorized use or (5) otherwise misuse this system are strictly prohibited. Such attempts or acts are subject to action that may result in criminal, civil, or administrative penalties.

[13] See https://vaww.ocis.va.gov/portal/server.pt?.

KAAJEE Configuration File (i.e., kaajeeConfig.xml)

Figure 6-2. Sample KAAJEE configuration file (i.e., kaajeeConfig.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<kaajee-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="kaajeeConfig.xsd">

  <!-- host application name, used for login page display and logging -->
  <host-application-name>KAAJEE Sample</host-application-name>

  <!-- put each station number for KAAJEE login here -->
  <login-station-numbers>
    <station-number>###</station-number>
    <station-number>###9XX</station-number>
    <station-number>###9XX</station-number>
    <station-number>###XX</station-number>
    <station-number>###XX</station-number>
    <station-number>###</station-number>
    <station-number>###9XX</station-number>
    <station-number>###9XX</station-number>
    <station-number>###XX</station-number>
    <station-number>###XX</station-number>
  </login-station-numbers>


  <!-- defined application context root Name -->
  <context-root-name>/kaajeeSampleApp</context-root-name>


  <!-- put the system announcement here. Use ~ for a line break, or ~ ~ for a
       paragraph break. -->
  <system-announcement>
    U.S. Government Computer System
    ~ ~
    U. S. government systems are intended to be used by authorized government
network users for viewing and retrieving information only, except as otherwise
explicitly authorized for official business and limited personal use in accordance
with policy.  Information from these systems resides on and transmits through
computer systems and networks funded by the government.  All access or use
constitutes understanding and acceptance that there is no reasonable expectation of
privacy in the use of Government networks or systems.
     ~ ~
    The data and documents on this system include Federal records that contain
sensitive information protected by various Federal statutes, including the Privacy
Act, 5 U.S.C. Section 552a, and veterans' records confidentiality statutes such as
38 U.S.C. Sections 5701 and 7332.  Access to the data and records is on a need-to-
know basis only.
    ~ ~
    All access or use of this system constitutes user understanding and acceptance
of these terms and constitutes unconditional consent to review and action including
(but not limited to) monitoring, recording, copying, auditing, inspecting,
investigating, restricting access, blocking, tracking, disclosing to authorized
personnel, or any other authorized actions by all authorized government and law
enforcement personnel.
     ~ ~
    Unauthorized user attempts or acts to (1) access, upload, change, or delete
information on this system, (2) modify this system, (3) deny access to this system,
(4) accrue resources for unauthorized use or (5) otherwise misuse this system are
strictly prohibited.  Such attempts or acts are subject to action that may result
in criminal, civil, or administrative penalties.
```

```
  </system-announcement>

  <!-- set to true to return a user's "New Person" division multiple as part
       of login -->
  <user-new-person-divisions retrieve="true" />

  <!-- set to true to return all children divisions of the login division's
       computing facility, as part of login -->
  <computing-facility-divisions retrieve="true" />

  <cactus-insecure-mode enabled="false" />

  <!-- BEAWeblogic Server SSL listen port , used for login page to implement SSL
-->
  <!--   <ssl-listen-port-number>7002</ssl-listen-port-number>  -->


</kaajee-config>
```

7. Programming Guidelines

Application Involvement in User/Role Management

Under ordinary circumstances, an application that is Kernel Authentication and Authorization Java (2) Enterprise Edition (KAAJEE)-enabled should not record, store, or otherwise manage which user divisions are legal for a user to log into, or which roles a user has been granted. Kernel acts as the external source of Authentication and Authorization, as well as the point of user provisioning.

With KAAJEE, the IRM system manager handles all these tasks on the VistA M Server. This is one of the benefits of the KAAJEE approach; the user and role administration is all handled at the same VistA M Server location as it always has been.

J2EE Container-enforced Security Interfaces

As with any security framework solution (e.g., SSPIs), all J2EE container-enforced security is supported. You can access the username of the end-user programmatically, and you can use both programmatic and declarative role checking to protect resources.

The web.xml and weblogic.xml files are used for declarative role checking. Using the isUserInRole and/or isCallerInRole methods are considered programmatic authorization/role checking. Using custom SSPIs with J2EE Form-based Authentication (e.g., KAAJEE) can be considered programmatic Authentication and Authorization. Using Basic Authentication with just deployment descriptors is purely declarative Authentication and Authorization. Whenever code is added to the equation of deciding Authentication and Authorization, then it becomes programmatic.

J2EE Username Format

For KAAJEE, the J2EE username for a given user is returned in the following format:

```
xxxx_DUZ_nnnn~CMPSYS_nnn
```

Where:

- **xxxx—**The first four characters following the "/" of the value as entered in the <context-root-name> tag in the kaajeeConfig.xml file.
- **DUZ_nnnn—**The user's DUZ as stored in the NEW PERSON file (#200).
- **CMPSYS_nnn—**The Station Number of the login division's computing system provider as returned by Standard Data Services' Institution getVistaProvider() API.

REF: For more information on the use of the SDS APIs, please refer to the *SDS API Installation Guide*. The SDS documentation is included in the SDS software distribution ZIP files, which are available for download at the following Website:

http://vaww.sts.infoshare.va.gov/STS_SDS/Project%20Artifacts/Forms/AllItems.aspx

For example:

```
kaaj_DUZ_8888~CMPSYS_523
```

Where:

- kaaj—The first four characters following the "/" of the value as entered in the <context-root-name> tag in the kaajeeConfig.xml file.
- 8888—The user's DUZ as stored in the NEW PERSON file (#200).
- 523—The Station Number of the login division's computing system provider, as returned by Standard Data Services' Institution getVistaProvider() API.

On the VistA M Server, this should correspond to the Station Number of the default Institution, as defined in the KAAJEE login host computer system's KERNEL SYSTEM PARAMETERS file (#8989.3).

This means that for all the divisions supported on a given VistA M Server, a user will have the same J2EE username returned to them. For logins against a different computer system, the same user will likely have a different DUZ, as well as a different parent facility, returned.

NOTE: In the future, the Department of Veterans Affairs Personal Identification (VPID) may alter the username, assuming an enterprise-wide user identifier is created in VHA or VA. The VPID will be stored in the NEW PERSON file (#200), in addition to being stored in national directories.

LoginUserInfoVO Object

After login, KAAJEE returns additional demographic information in a LoginUserInfoVO object(i.e., value object). KAAJEE stores the LoginUserInfoVO object (i.e., value object) in the Hyper Text Transport Protocol (HTTP) Session Object. The object is stored in the session object using the key value stored in the LoginUserInfoVO.SESSION_KEY string.

LoginUserInfoVO is implemented as a JavaBean, therefore it can be accessed as a JavaBean, within Java Server Pages (JSP) Web pages.

NOTE: A JavaBean is a reusable component that can be used in any Java application development environment. JavaBeans are dropped into an application container, such as a form, and can perform functions ranging from a simple animation to complex calculations.[14]

[14] Definition of JavaBean from the following Glossary Website: http://www.orafaq.com/glossary/faqglosj.htm, 7/17/04, Revision 2.1; Author: Frank Naudé.

For example:

Figure 7-1. JavaBean Example: LoginUserInfoVO object

```
public class LoginUserInfoVO
extends java.lang.Object
implements java.io.Serializable
```

KAAJEE returns this JavaBean to the enclosing application after login. It is returned to the enclosing application as an object in HttpSession. It contains user demographics information about the logged-in user. A public static field provides the key for the application to find the object in HttpSession.

Table 7-1. Field Summary: LoginUserInfoVO object

Field Summary	
static java.lang.String	**SESSION_KEY** The key under which this value is placed in the session object during login, and from which this object can be retrieved by the enclosing Web-based application post-login.

Table 7-2. Constructor Summary: LoginUserInfoVO object

Constructor Summary
LoginUserInfoVO() generic constructor.

Table 7-3. Method Summary: LoginUserInfoVO object

Method Summary	
Return Type	**Method Name and Description**
java.util.TreeMap	**getLoginDivisionVistaProviderDivisions()** Returns a list of divisions (based on information in the SDS Institution table) whose Vista Provider is the same as the Vista Provider computer system of the login division. This list is returned as a TreeMap. The key value in the TreeMap is the Station Number, which is a String.The object value stored under each key is a VistaDivisionVO object. **REF:** See also the "VistaDivisionVO Object" topic in this manual. This method is provided to applications to support division switching for all divisions supported at the same computing facility as the login division, regardless of whether explicit access has been granted to the user for any particular division. Applications can display a list of other divisions that the user could switch to within the application, allowing the user to select a different division. It is then the application's responsibility to use the proper division for its own internal business rules. The application developer should be aware that this method may not be appropriate when using VistALink RPC calls as the login user may not be permitted access to a specific division.
java.lang.String	**getLoginStationNumber**() Returns the Station Number of the Division the user selected at login. This can be used as a key to retrieve additional information (e.g., name about the login division from the TreeMap of permitted divisions returned by the getPermittedDivisions method).
java.util.TreeMap	**getPermittedNewPersonFileDivisions**() Returns a list of the user's permitted divisions returned as a TreeMap. The key value in the TreeMap is the Station Number, which is a String. The object value stored under each key is a VistaDivisionVO object. **REF:** See also the "VistaDivisionVO Object" topic in this manual. This list represents all of the divisions on the VistA M Server that the user could have logged into. Applications can display a list of other divisions that the user could switch to within the application, allowing the user to select a different division. It is then the application's responsibility to use the proper division for its own internal business rules, and also to pass the proper Division Station Number with each VistALink RPC call it makes to M.
java.lang.String	**getUserDegree**() Returns the user's Degree value from the NAME COMPONENTS file (#20).
java.lang.String	**getUserDuz**() Return the user's DUZ from the NEW PERSON file (#200).

Method Summary	
Return Type	**Method Name and Description**
java.lang.String	**getUserFirstName**() Returns the users' First Name value from the NAME COMPONENTS file (#20).
java.lang.String	**getUserLastName**() Returns the user's Last Name value from the NAME COMPONENTS file (#20).
java.lang.String	**getUserMiddleName**() Returns the user's Middle Name value from the NAME COMPONENTS file (#20).
java.lang.String	**getUserName01**() Returns the user's name as it's stored in the NAME field (# .01) in the NEW PERSON file (#200). For example: `KRNUSER,ONE E`
java.lang.String	**getUserNameDisplay**() Returns the Display Name of the user, as put together by the Name Standardization APIs on M. For example: `One E. Krnuser`
java.lang.String	**getUserParentAdministrativeFacilityStationNumber**() Returns the parent facility of the Division used for login, as resolved on the login computer system based on that system's INSTITUTION file (#4) from the SDS 3.0 (or higher) tables.
java.lang.String	**getUserParentComputerSystemStationNumber**() Returns the computer system's default Institution/Computer System Institution, as identified in the system's KERNEL SYSTEM PARAMETERS file (#8989.3).
java.lang.String	**getUserPrefix**() Returns the user's Prefix value from the NAME COMPONENTS file (#20).
java.lang.String	**getUserSuffix**() Returns the user's Suffix value from the NAME COMPONENTS file (#20).
java.lang.String	**toString**() Returns a string representation of the values in the object.

An example of using this JavaBean in a Java Server Page (JSP) Web page is shown below:

Figure 7-2. Sample JSP Web page code (e.g., AppHelloWorld.jsp)

```
<%@ page language="java" %>
<%@ page import ="gov.va.med.authentication.kernel.LoginUserInfoVO,
 gov.va.med.authentication.kernel.VistaDivisionVO,
 java.util.Set,
 java.util.Iterator,
 java.util.TreeMap,
 javax.naming.NamingException,
 javax.resource.ResourceException;" %>

<html>
<head><title>Hello, World</title></head>
<body>
  <% String groupname = "XUKAAJEE_SAMPLE_ROLE";
  %>

<h2>Hi there. This web page is a protected application resource.</h2>
<h2>[YOUR APP PAGE GOES HERE]</h2>

<p><i>To get here you needed to both <i>authenticate</i> and <i>authorize</i>.<br>
  So let's see who you are.</i></p>

<p><b>Authenticated username -- request.getRemoteUser(): </b><font color="red"><%=
request.getRemoteUser() %>
  </font></p>

<p><b>Authorization -- request.isUserInRole(&amp;quot;<%= groupname %>&amp;quot;)?:
</b><font color="red">
  <%= request.isUserInRole(groupname) %></font> <br>
<b>Authorization -- request.isUserInRole(AUTHENTICATED_KAAJEE_USER)?: </b><font
color="red">
  <%= request.isUserInRole("AUTHENTICATED_KAAJEE_USER") %></font><br>


<b>Authorization -- request.principal name ?: </b><font color="red">
  <%= request.getUserPrincipal() %></font><br>


  <% LoginUserInfoVO userLoginInfo =
     (LoginUserInfoVO) session.getAttribute(LoginUserInfoVO.SESSION_KEY);
     pageContext.setAttribute("userInfo", userLoginInfo);
  %>
  <jsp:useBean id="userInfo" scope="page"
   type="gov.va.med.authentication.kernel.LoginUserInfoVO" />
  <table border="0" cellspacing="3" cellpadding="3">

  <tr align="left">
    <td colspan="2">
      <p><strong>User Info (from Session): </strong></p></td>
    </tr>
    <tr>
      <td align="right"><b>DUZ:</b></td>
      <td><jsp:getProperty name="userInfo" property="UserDuz" /></td>
    </tr>
    <tr>
      <td align="right"><b>User name (.01 New Person): </b></td>
      <td><jsp:getProperty name="userInfo" property="UserName01" /></td>
    </tr>
```

```
    <tr>
      <td align="right"><b>User name (display):</b></td>
      <td><jsp:getProperty name="userInfo"
           property="UserNameDisplay" /></td>
    </tr>
    <tr>
      <td align="right"><b>Last Name:</b></td>
      <td><jsp:getProperty name="userInfo"
           property="UserLastName" /></td>
    </tr>
    <tr>
      <td align="right"><b>First Name:</b></td>
      <td><jsp:getProperty name="userInfo"
           property="UserFirstName" /></td>
    </tr>
    <tr>
      <td align="right"><b>Middle name:</b></td>
      <td><jsp:getProperty name="userInfo"
           property="UserMiddleName" /></td>
    </tr>
    <tr>
      <td align="right"><b>Prefix:</b></td>
      <td><jsp:getProperty name="userInfo" property="UserPrefix" /></td>
    </tr>
    <tr>
      <td align="right"><b>Suffix:</b></td>
      <td><jsp:getProperty name="userInfo" property="UserSuffix" /></td>
    </tr>
    <tr>
      <td align="right"><b>Degree:</b></td>
      <td><jsp:getProperty name="userInfo" property="UserDegree" /></td>
    </tr>
    <tr>
      <td align="right"><b>Login Station Number:</b></td>
      <td><jsp:getProperty name="userInfo"
           property="LoginStationNumber" /></td>
    </tr>
    <tr>
      <td align="right"><b>Parent Administrative
        Facility Station Number:</b></td>
      <td><jsp:getProperty name="userInfo"
        property="UserParentAdministrativeFacilityStationNumber" /></td>
    </tr>
    <tr>
      <td align="right"><b>Parent Computer System Station Number:</b></td>
      <td><jsp:getProperty name="userInfo"
        property="UserParentComputerSystemStationNumber" /></td>
    </tr>
    <tr>
      <td align="right" valign="top"><b>Permissible Divisions
        (New Person file):</b></td>
      <td>
        <%
        StringBuffer sb = new StringBuffer();
        {
          TreeMap permittedDivisions =
             userLoginInfo.getPermittedNewPersonFileDivisions();
          if (permittedDivisions != null) {
            Set keySet = permittedDivisions.keySet();
            Iterator it = keySet.iterator();
            while (it.hasNext()) {
              String divNumber = (String) it.next();
              VistaDivisionVO vDiv =
```

```
                (VistaDivisionVO) permittedDivisions.get(divNumber);
              sb.append(vDiv.toString());
              sb.append("<br>");
            }
          }
        }
        %>
        <%= sb.toString() %>
      </td>
    </tr>
    <tr>
      <td align="right" valign="top">
      <b>Divisions that are children of
      <br>the Login Division's Computing Facility
      <br>institution, sharing the same computing
      <br>facility:</b></td>
      <td>
        <%
        sb = new StringBuffer();
        {
          TreeMap cfDivisions =
            userLoginInfo.getLoginDivisionVistaProviderDivisions();
          if (cfDivisions != null) {
            Set keySet = cfDivisions.keySet();
            Iterator it = keySet.iterator();
            while (it.hasNext()) {
              String divNumber = (String) it.next();
              VistaDivisionVO vDiv =
                (VistaDivisionVO) cfDivisions.get(divNumber);
              sb.append(vDiv.toString());
              sb.append("<br>");
            }
          }
        }
        %>
        <%= sb.toString() %>
      </td>
    </tr>
  </table>
  <p><a href="logout.jsp"><b>LOGOUT</b></a></p>
</body>
</html>
```

VistaDivisionVO Object

The VistaDivisionVO object JavaBean is used to store an individual division, when division TreeMaps (i.e., tree structure, keyed on Division Station Number strings) are returned by the LoginUserInfoVO methods.

REF: For more information on the LoginUserInfoVO methods, please refer to Table 7-3 in this chapter.

For example:

Figure 7-3. JavaBean Example: VistaDivisionVO object

```
public class VistaDivisionVO
extends java.lang.Object
implements java.io.Serializable

Represents a VistA Division, including Station Name and Station Number.
```

Table 7-4. Constructor Summary: VistaDivisionVO object

Constructor Summary
VistaDivisionVO() Instantiates a VistaDivision with all fields set to a null string.

Table 7-5. Method Summary: VistaDivisionVO object

Method Summary	
Return Type	**Method Name and Description**
boolean	**getIsDefault**() Returns whether or not this is set to the default Login Division.
java.lang.String	**getName**() Returns the Station Name of the Division, presumably from the VistA M Server INSTITUTION file (#4) entry (depending on the source of the information the instance contains)
java.lang.String	**getNumber**() Returns the Station Number of the Division, presumably from the VistA M Server INSTITUTION file (#4) entry (depending on the source of the information the instance contains)
java.lang.String	**toString**() Returns a string representation of the Division information

VistALink Connection Specs for Subsequent VistALink Calls

For subsequent VistALink calls (i.e., after the user has already been authenticated), application developers can use one of the VistALink connection specs for general application use. The information returned by the KAAJEE login helps streamline this process.

For example, if your J2EE application needs to make a VistALink connection to the same division under which the user logged in (a frequent circumstance for some applications), application developers can use the VistaLinkDuzConnectionSpec. This connection spec identifies the user to the VistA M Server based on the user's DUZ (i.e., Kernel user internal entry number [IEN]) in the NEW PERSON file (#200).

Thus, for subsequent VistALink calls, an application can do any of the following:

- Retrieve the division against which the user logged in from the LoginUserInfoVO object.
- Retrieve the JNDI name for the corresponding VistALink connector pool using the Login Division.

  The JNDI can be retrieved by using VistALink's InstitutionMappingDelegate.getJndiConnectorNameForInstitution method. The following are examples of the usage of this method:

  ```
  String jndiConnectionName =
  InstitutionMappingDelegate.getJndiConnectorNameForInstitution(institution);

  String jndiName =
  InstitutionMappingDelegate.getJndiConnectorNameForInstitution(division);
  ```

- Retrieve the user's DUZ from the LoginUserInfoVO object.
- Make the connection to the VistA M Server using the VistaLinkDuzConnectionSpec. This particular connection specification class does not require any additional user mapping on the VistA M Server/Kernel side. As long as there is a "trust" relationship between your J2EE Application Server and the VistA M Server in question, then there should be no reason not to use the VistaLinkDuzConnectionSpec.

REF: For more information on the LoginUserInfoVO object, please refer to the "LoginUserInfoVO Object" topic in this chapter.

NOTE: The VistaLinkDuzConnectionSpec has been deprecated; however, its use will most likely continue until the conversion to VPIDs is completed.

REF: For more information on the VistALink connection specs, please refer to the *VistALink Developer Guide (Version 1.5)*.

Providing the Ability for the User to Switch Divisions

Applications that support multi-divisional functionality need to manage the set of divisions between which a user can switch. KAAJEE supports this need by providing valid lists of divisions to which the user can switch.

KAAJEE provides two different division lists, because different applications have different business rules as to which divisions should be supported:

- Divisions from a User's New Person File
- All Divisions at the Login Division's Computing Facility

Divisions from a User's New Person File

Some applications want to support division switching only to those divisions that an IRM system manager has configured as valid divisions in a user's NEW PERSON file (#200) entry on their host VistA M Server. To obtain this list of divisions from KAAJEE:

1. Configure the KAAJEE software to retrieve this information. In the kaajeeConfig.xml file, set the following tag to "true" (case sensitive):

   ```
   <user-new-person-divisions retrieve="true" />
   ```

2. Access the list in the LoginUserInfoVO object, using the getPermittedNewPersonFileDivisions() method.

The list of divisions from the user's DIVISION Multiple field (#16) in the NEW PERSON file (#200) on the VistA M Server is filtered. The DIVISION *must* be within the same computing facility as the KAAJEE Login Division, as determined by the Standard Data Services (SDS) Institution utilities (i.e., Institution.getVistaProvider method).

All Divisions at the Login Division's Computing Facility

Some applications want to support division switching for all divisions supported at the same computing facility as the login division, regardless of whether explicit access has been granted to the user for any particular division. To obtain this list of divisions from KAAJEE do the following:

1. Configure the KAAJEE software to retrieve this information. In the kaajeeConfig.xml file, set the following tag to "true" (case sensitive):

   ```
   <computing-facility-divisions retrieve="true" />
   ```

2. Access the list in the LoginUserInfoVO object using the getLoginDivisionVistaProviderDivisions() method.

The list of divisions is filtered. Divisions *must* be within the same computing facility as the KAAJEE Login Division, as determined by the SDS Institution utilities (i.e., Institution.getVistaProvider method).

logout.jsp File

The KAAJEE listeners (see Table 4-6) listen for session logouts. Logouts can ether be user-initiated or due to a session timeout. If a logout is detected (i.e., session.invalidate), the KAAJEE listeners call the XUS KAAJEE LOGOUT RPC (see Table 8-1.) to log the user off of the system and update the SIGN-ON LOG file (#3.081) to show the user is now logged off of the system.

REF: For more information on the SIGN-ON LOG file (#3.081), please refer to the *Kernel Systems Management Guide*.

KAAJEE 1.0.1.xxx distributes a sample logout.jsp file, it is located in the following directory:

<STAGING_FOLDER>/kaajee-1.0.1.xxx/jars/jsp/logout.jsp

The sample logout.jsp file is shown below:

Figure 7-4. Sample logout.jsp file

```
<%@ page language="java" %>
<HTML>
  <HEAD>
    <!--
     *
     * @author Security Service
     * @version 1.0.1.xxx
     * -->
    <TITLE>Logout Page</TITLE>
  </HEAD>
  <BODY>
    <%
      session.invalidate();
    %>
    <H3>You are now logged out.</H3>
  </BODY>
</HTML>
```

This sample logout.jsp file is an optional and is only provided as a template on how to provide a logout link and corresponding logout.jsp. However, consuming applications *must* provide a means for the user logged in to log out.

For example, to support logout of a Web application's protected resource, the Web application would need to provide an HTML link to call the logout.jsp (see Figure 7-4). The HTML code for such a link would look similar to the following:

Figure 7-5. Sample HTML code to call the logout.jsp file

```
<P><A HREF="logout.jsp"><B>LOGOUT</B></A></P>
```

III. Systems Management Guide

This is the Systems Management Guide section of this supplemental documentation for Kernel Authentication and Authorization Java (2) Enterprise Edition (KAAJEE). It is intended for use in conjunction with the KAAJEE software. It details the technical-related KAAJEE documentation (e.g., implementation and maintenance of KAAJEE, routines, files, options, interfaces, product security, etc.).

8. Implementation and Maintenance (J2EE Site)

Information throughout this manual is meant to help IRM in the implementation and maintenance of Kernel Authentication and Authorization Java (2) Enterprise Edition (KAAJEE).

Namespace

KAAJEE consists of VistA M Server patches that have been assigned to the following namespaces (listed alphabetically):

- XU—Kernel
- XWB—RPC Broker

NOTE: Kernel is the designated custodial software application for KAAJEE; however, KAAJEE comprises multiple patches and software releases from several Health***e***Vet-VistA applications.

REF: For the specific KAAJEE software and VistA M Server patches required for the implementation of KAAJEE, please refer to Table 1-2 in the "KAAJEE" topic in Chapter 1 in this manual.

KAAJEE SSPI Tables—Deleting Entries

KAAJEE does not currently purge the two KAAJEE SSPI tables at system startup. It only deletes and recreates individual user entries in the tables during the login process.

REF: For more information regarding the KAAJEE SSPI tables, please refer to the *KAAJEE Installation Guide*.

KAAJEE Login Server Requirements

In a domain consisting of an Administration Server and several Managed Servers, the Administration Server *must* always be running, as new logins through KAAJEE will *not* succeed while the Administration Server is down.

Log4J Configuration

In order to provide a unified logger and consolidate all log/error entries into one file, all J2EE-based application-specific loggers *must* be added to the same log4j configuration file, which should be the active log4j configuration file for the server. After locating the active log4j configuration file used on the server you are configuring (e.g., mylog4j.xml file), add in the KAAJEE (and FatKAAT) loggers to that file.

To locate the active log4j configuration file, look for the"-Dlog4j.configuration=" argument in the startup script file (i.e., startWebLogic.sh or startWebLogic.cmd). The "-Dlog4j.configuration=" should be set to the absolute location of the configuration file (e.g., c:/mydirectory/mylog4j.xml). If no such argument is present, look for a file named "log4j.xml" in a folder on the server classpath.

You *must* configure log4j for the first time, if all three of the following conditions exist:

- The "-Dlog4j.configuration=" argument does *not* exist in the WebLogic JVM startup script files.
- The "log4j.xml" file does *not* exist in the classpath.
- There is no pre-existing log4j configuration file in the folder placed on the classpath of the WebLogic Application Server containing the configuration files for all Health***e***Vet-VistA J2EE applications (e.g., <HEV CONFIGURATION FOLDER>).

For first time log4j configuration procedures, please refer to the "log4j Configuration File" topic in the *VistALink Installation Guide*. Also, sample log4j configuration files are included with the VistALink 1.5 software distribution.

REF: For more information on VistALink, please refer to the VistALink documentation located on the VHA Software Document Library (VDL) Website at the following Website:

http://www.va.gov/vdl/application.asp?appid=163

Once the log4j file is initially configured, you need to configure the file specifically for KAAJEE log entries as outlined in the *KAAJEE Installation Guide*.

REF: For the specific step-by-step procedures on how to configure the log4j for KAAJEE, please refer to the "Configure log4j for All J2EE-based Application Log Entries" topic in the *KAAJEE Installation Guide*.

REF: For more information on log4j guidelines, please refer to the Application Structure & Integration Services (ASIS) *Log4j Guidelines for Health**e**Vet-VistA Applications* document available at the following Website:

http://vista.med.va.gov/vistaarch/healthevet/Documents/Log4j%20Guidance%20v1.0.doc

Log Monitoring

Log4J Log

In test, developers use this log during Web application development as a debugging tool. It can provide detailed context for application and authentication failures. It is a complimentary tool for testing applications.

In production, Web administrators should monitor this log. If a problem is detected and developers or the Web administrators are unable to resolve it, the user should call the National Help Desk and file a Remedy ticket.

The following figure (Figure 8-1) shows sample data in the log4j file:

Figure 8-1: Sample logout log4j.xml file entries

```
.
.
.
1221895406 2006-05-02 14:51:53,252 [ExecuteThread: '14' for queue:
'weblogic.kernel.Default'] DEBUG
gov.va.med.authentication.kernel.KaajeeHttpSessionListener:sessionDestroyed:41 -
Session destroyed GXSXGMQCLDx3STLtSLNMQ1zZSGzLSfsBZ9Dsf6p1hmTTGNz7S76l!-
1114227413!1146606295311
1221895406 2006-05-02 14:51:53,252 [ExecuteThread: '14' for queue:
'weblogic.kernel.Default'] DEBUG
gov.va.med.authentication.kernel.KaajeeHttpSessionListener:sessionDestroyed:46 -
Got LoginUserInfoVO object.
        matchManagedConnection-
>gov.va.med.vistalink.adapter.spi.VistaLinkManagedConnection[]10.6.21.15[]18001[]1[
]J2EE[fdi]2[mdi]1->gov.va.med.vistalink.adapter.spi.VistaLinkConnectionRequestInfo-
>gov.va.med.authentication.kernel.KaajeeVistaLinkConnectionSpec->av
        matchManagedConnection->no match on request info-
>gov.va.med.vistalink.adapter.spi.VistaLinkManagedConnection[]10.6.21.15[]18001[]1[
]J2EE[fdi]2[mdi]1->gov.va.med.vistalink.adapter.spi.VistaLinkConnectionRequestInfo-
>gov.va.med.authentication.kernel.KaajeeVistaLinkConnectionSpec->av
        Connection re-authentication status: 'notauthenticated'. cri =
gov.va.med.vistalink.adapter.spi.VistaLinkConnectionRequestInfo-
>gov.va.med.authentication.kernel.KaajeeVistaLinkConnectionSpec->av
1221895419 2006-05-02 14:51:53,265 [ExecuteThread: '14' for queue:
'weblogic.kernel.Default'] DEBUG
gov.va.med.authentication.kernel.LoginControllerUtils:getVistaLinkConnection:178 -
got connection.
1221895447 2006-05-02 14:51:53,293 [ExecuteThread: '14' for queue:
'weblogic.kernel.Default'] DEBUG
gov.va.med.authentication.kernel.LogoutController:performLogoutActions:74 -
Executed RPC to mark signon log at station #'662BU' for user DUZ '1000098' logged
off for signon log IEN '3060502.144534'.
1221895450 2006-05-02 14:51:53,296 [ExecuteThread: '14' for queue:
'weblogic.kernel.Default'] DEBUG
gov.va.med.authentication.kernel.KaajeeSessionAttributeListener:attributeRemoved:42
- Attribute removed: gov.va.med.authentication.kernel.LoginUserInfo
1221895451 2006-05-02 14:51:53,297 [ExecuteThread: '14' for queue:
'weblogic.kernel.Default'] DEBUG
gov.va.med.authentication.kernel.KaajeeSessionAttributeListener:attributeRemoved:47
- Found LoginUserInfoVO object.
1221895464 2006-05-02 14:51:53,310 [ExecuteThread: '14' for queue:
'weblogic.kernel.Default'] DEBUG
gov.va.med.authentication.kernel.LoginControllerUtils:getVistaLinkConnection:178 -
got connection.
1221895476 2006-05-02 14:51:53,322 [ExecuteThread: '14' for queue:
'weblogic.kernel.Default'] DEBUG
gov.va.med.authentication.kernel.LogoutController:performLogoutActions:74 -
Executed RPC to mark signon log at station #'662BU' for user DUZ '1000098' logged
off for signon log IEN '3060502.144534'.
.
.
.
```

In the sample log entries above (Figure 8-1), only the KAAJEE-specific logout-related entries are displayed, the VistALink entries have been filtered out. If included, the VistALink entries would show the "about to execute RPC:" and the "Completed execution of RPC: 'XUS KAAJEE LOGOUT'."

M-side Log

This event log records VistA M Server-related errors. IRM should monitor this log for any errors related to KAAJEE and take appropriate actions to remedy the error.

Sign-On Log

This event log records all users that sign onto the VistA M Server via Kernel in the SIGN-ON LOG file (#3.081)**.** IRM should monitor this log. IRM should check for unusual activity (e.g., unusual amount of activity for a given user). If there is an unusual amount of activity for a particular user, IRM should further investigate by contacting the user in question and taking appropriate action as deemed appropriate.

REF: For more information on the SIGN-ON LOG file (#3.081), please refer to the *Kernel Systems Management Guide*.

Failed Access Attempts Log

This event log records users that fail to enter a valid Access/Verify code pair**.** IRM should monitor this log and check for unusual activity (e.g., unusual amount of activity for a given user). If there is an unusual amount of activity for a particular user, IRM should further investigate by contacting the user in question and taking appropriate action as deemed appropriate.

Remote Procedure Calls (RPCs)

The following remote procedure calls (RPC) are exported with KAAJEE (listed alphabetically):

Table 8-1. KAAJEE-related RPC list

RPC Name	RPC Description
XUS ALLKEYS	Kernel Patch XU*8.0*329 exports this RPC. This RPC returns all J2EE VistA M Server J2EE security keys (i.e., those security keys with the SEND TO J2EE field [#.05] in the SECURITY KEY file [#19.1] set to YES).
XUS KAAJEE GET USER INFO	Kernel Patch XU*8.0*329 exports this RPC. This RPC returns a variety of user demographics and other information (e.g. DUZ, user name, degree, Station Numbers, etc.) needed for users to sign onto the VistA M Server via KAAJEE. It returns the following in the results array. RESULT(0)—User's DUZ from the NEW PERSON file (#200).

RPC Name	RPC Description
	RESULT(1)—User name from the .01 field of the NEW PERSON file (#200). RESULT(2)—User's full name from the NAME COMPONENTS file (#20). RESULT(3)—FAMILY (LAST) NAME from the NAME COMPONENTS file (#20). RESULT(4)—GIVEN (FIRST) NAME from the NAME COMPONENTS file (#20). RESULT(5)—MIDDLE NAME from the NAME COMPONENTS file (#20). RESULT(6)—PREFIX from the NAME COMPONENTS file (#20). RESULT(7)—SUFFIX from the NAME COMPONENTS file (#20). RESULT(8)—DEGREE from the NAME COMPONENTS file (#20). RESULT(9)—Station Number of the division in which the user is working. RESULT(10)—Station Number of the parent facility for the login division from the INSTITUTION file (#4). RESULT(11)—Station Number of the parent "computer system" from the KERNEL SITE PARAMETERS file (#8989.3). RESULT(12)—Signon log entry IEN. RESULT(13)—Number of permissible divisions. RESULT(14 - n)—Permissible divisions for user login, in the following format: IEN of file 4^Station Name^Station Number^default? (1 or 0)
XUS KAAJEE LOGOUT	Kernel Patch XU*8.0*329 exports this RPC. This RPC calls the LOUT^XUSCLEAN API in order to mark a KAAJEE-signed on user's entry in the SIGN-ON LOG file (#3.081) as signed off.

REF: For more information on these RPCs, please refer to the REMOTE PROCEDURE file (#8994) or the Kernel RPC Website located at the following Website:

http://vista.med.va.gov/kernel/rpcs/index.shtml

Files and Fields

There are *no* new VistA M Server files or fields *directly* exported with KAAJEE; however, the following modified file and new field are *associated with* KAAJEE and exported with Kernel Patch XU*8.0*337:

Table 8-2. KAAJEE-related software new fields

File Number	File Name	Field Name	Field Number	Field Description
19.1	SECURITY KEY	SEND TO J2EE	.05	This field was released with Kernel Patch XU*8.0*337. It indicates whether or not a VistA M Server security key is a J2EE-related security key and should be sent to the application server for temporary role assignment. Application developers *must* set this field to YES for those security keys that correspond to WebLogic group names that are stored in the application's weblogic.xml file. **REF:** For more information on J2EE security–related keys and WebLogic groups, please refer to "2.Create VistA M Server J2EE security keys Corresponding to WebLogic Group Names" topic in Chapter 5, "Role Design/Setup/Administration," in this manual.

Global Mapping/Translation, Journaling, and Protection

There are *no* special global mapping/translation, journaling, and protection instructions for KAAJEE.

Routine(s)

This topic contains a list of the new VistA M Server routine exported with KAAJEE. A brief description of the routine is provided.

Table 8-3. KAAJEE-related software routine list

Routine Name	Routine Description
XUSKAAJ	This is the KAAJEE routine.

Exported Options

The following menu options are exported with KAAJEE (listed alphabetically):

Table 8-4. KAAJEE exported options

Option Name	Option Description
XUCOMMAND	This menu option is used to link the XUS KAAJEE WEB LOGON option. As all authenticated users have access to XUCOMMAND, this linkage enables all users to have access to all RPCs listed under the XUS KAAJEE LOGON "B"-type option.
XUS KAAJEE WEB LOGON	This "B"-type option contains references to the following RPCs in its "RPC" multiple: • XUS ALLKEYS • XUS KAAJEE GET USER INFO • XUS KAAJEE LOGOUT This option has no effect on those RPCs as such; however, having this option assigned allows KAAJEE to call these RPCs on behalf of the end-user.

REF: For more information on KAAJEE-related RPCs, please refer to the "Remote Procedure Calls (RPCs)" topic in this chapter.

Archiving and Purging

There are *no* special archiving, purging, or journaling instructions for KAAJEE.

REF: For more information regarding the KAAJEE SSPI tables, please refer to the *KAAJEE Installation Guide*.

Callable Routines

There are *no* callable VistA M Server routines exported with KAAJEE.

External Relations

Health*e*Vet-VistA Software Requirements

KAAJEE relies on the following Health*e*Vet-VistA software to run effectively (listed alphabetically):

Table 8-5. External Relations—Health*e*Vet-VistA software

Software	Version	Description
Kernel	8.0	Server software—Fully patched.
Kernel Toolkit	7.3	Server software—Fully patched.
RPC Broker	1.1	Client/Server software—Fully patched.
Standard Data Services (SDS)	3.0 (or higher)	Database and Software—Fully patched. Contains Institution-related data tables accessed via supported APIs created by SDS. **NOTE:** KAAJEE works with SDS 3.0 or higher; however, KAAJEE 1.0.1.xxx distributes SDS 13.0 client jar files as part of the Sample Web Application. If you deploy the both the KAAJEE Sample Web Application and your own Web-based application on the same WebLogic Application Server domain instance and intend to use a different version of SDS, those client jar files will need to be swapped out for the appropriate version of the SDS client jar files. Otherwise, There may be a conflict if both applications reference the same JNDI tree.
VA FileMan	22.0	Server software—Fully patched.
VistALink	1.5	Client/Server software—Fully patched.

NOTE: Kernel is the designated custodial software application for KAAJEE; however, KAAJEE comprises multiple patches and software releases from several Health*e*Vet-VistA applications.

REF: For the specific KAAJEE software and VistA M Server patches required for the implementation of KAAJEE, please refer to Table 1-2 in the "KAAJEE" topic in Chapter 1 in this manual.

COTS Software Requirements

The KAAJEE authorization and authentication software interface with the following Commercial-Off-The-Shelf (COTS) software products in order to run effectively (listed alphabetically):

Table 8-6. External Relations—COTS software

Software	Version	Description
WebLogic	8.1 (SP4 or higher)	Application server software—Fully patched.
Java IDE (e.g., MyEclipse/ Eclipse)	Any	Developer workstation software—The Java Integrated Development Environment (IDE) is used when developing J2EE Web-based applications that are KAAJEE-enabled.
Java 2 Standard Edition (J2SE) Java Development Kit (JDK, e.g., Sun Microsystems')	Any	Developer workstation software—Fully patched. The JDK is used when developing J2EE Web-based applications that are KAAJEE-enabled. The JDK should include Java Runtime Environment (JRE) and other developer tools to write Java code.
Sentillion Web Software Development Kit (SDK)	TBD	Developer workstation software—The SDK is used when developing CCOW-aware and KAAJEE-enabled applications.

NOTE: There are *no* other COTS (*non*-VA) products embedded in or requiring special interfaces by this version of KAAJEE, other than those provided by the underlying operating systems.

DBA Approvals and Database Integration Agreements

The Database Administrator (DBA) maintains a list of Integration Agreements (IAs) or mutual agreements between software developers allowing the use of internal entry points or other software-specific features that are not available to the general programming public. These IAs are listed on FORUM.

KAAJEE is *not* dependent on any IAs; however, Kernel is the custodial package of KAAJEE Integration Agreement (IA) #4851.

To obtain the current list of IAs, if any, to which the Kernel (KAAJEE-related) software is a custodian:

1. Sign on to the FORUM system (forum.va.gov).
2. Go to the DBA menu [DBA].
3. Select the Integration Agreements Menu option [DBA IA ISC].
4. Select the Custodial Package Menu option [DBA IA CUSTODIAL MENU].
5. Choose the ACTIVE by Custodial Package option [DBA IA CUSTODIAL].
6. When this option prompts you for a package, enter **XXXX—**Where **XXXX** equals: **XU** or **Kernel**.
7. All current IAs to which the software is a custodian are listed.

To obtain detailed information on a specific integration agreement:

1. Sign on to the FORUM system (forum.va.gov).
2. Go to the DBA menu [DBA].
3. Select the Integration Agreements Menu option [DBA IA ISC].
4. Select the Inquire option [DBA IA INQUIRY].
5. When prompted for "INTEGRATION REFERENCES," enter the specific integration agreement number of the IA you would like to display.
6. The option then lists the full text of the IA you requested.

To obtain the current list of IAs, if any, to which the Kernel (KAAJEE-related) software is a subscriber:

1. Sign on to the FORUM system (forum.va.gov).
2. Go to the DBA menu [DBA].
3. Select the Integration Agreements Menu option [DBA IA ISC].
4. Select the Subscriber Package Menu option [DBA IA SUBSCRIBER MENU].
5. Choose the Print ACTIVE by Subscribing Package option [DBA IA SUBSCRIBER].

6. When prompted with "START WITH SUBSCRIBING PACKAGE," enter **XXXX** (in uppercase). When prompted with "GO TO SUBSCRIBING PACKAGE," enter **XXXX** (in uppercase)—Where "**XXXX**" equals: **XU**.
7. All current IAs to which the software is a subscriber are listed.

Internal Relations

Relationship of KAAJEE with the VistA M Server

Namespace

KAAJEE consists of VistA M Server patches that have been assigned to the following namespaces (listed alphabetically):

- XU—Kernel
- XWB—RPC Broker

NOTE: Kernel is the designated custodial software application for KAAJEE; however, KAAJEE comprises multiple patches and software releases from several Health*e*Vet-VistA applications.

REF: For the specific KAAJEE software and VistA M Server patches required for the implementation of KAAJEE, please refer to Table 1-2 in the "KAAJEE" topic in Chapter 1 in this manual.

Kernel 8.0

In order to develop J2EE Web-based applications so that they can be authorized and authenticated against Kernel, the following Kernel patches *must* be installed (listed in patch number order):

- Server Patch—XU*8.0*265
- Server Patch—XU*8.0*329
- Server Patch—XU*8.0*337

  NOTE: This patch is dependent on Kernel Patch XU*8.0*265.

- Server Patch—XU*8.0*361

  NOTE: This patch is not directly required by KAAJEE; however, since VistALink requires this patch and KAAJEE requires VistALink, this patch is included here.

- Server Patch—XU*8.0*430
- Server Patch—XU*8.0*451

RPC Broker 1.1

In order to develop J2EE Web-based applications so that they can be authorized and authenticated against Kernel, RPC Broker Patch 1*35 *must* be installed.

VistALink 1.5

In order to develop J2EE Web-based applications so that they can be authorized and authenticated against Kernel, VistALink 1.5 software (i.e., XOBS 1.5; fully patched) *must* be installed on the developer workstation and the application server.

Software-wide and Key Variables

KAAJEE does *not* employ the use of software-wide or key variables on the VistA M Server.

SACC Exemptions

KAAJEE does *not* have any Programming Standards and Conventions (SAC) exemptions.

9. Software Product Security

Security Management

There are *no* special legal requirements involved in the use of Kernel Authentication and Authorization Java (2) Enterprise Edition (KAAJEE).

Mail Groups, Alerts, and Bulletins

Mail Groups

KAAJEE does *not* create or utilize any specific mail groups.

Alerts

KAAJEE does *not* make use of alerts.

Bulletins

KAAJEE does *not* make use of bulletins.

Auditing—Log Monitoring

Log4J Log

In test, developers use this log during Web application development as a debugging tool. It can provide detailed context for application failures. It is a complimentary tool for testing applications.

In production, the Enterprise Management Center (EMC) and/or Application Server Administrators should monitor this log. If a problem is detected and developers or the administrators are unable to resolve it, the user should call the National Help Desk and file a Remedy ticket.

M-side Log

This event log records VistA M Server-related errors. Information Resource Management (IRM) should monitor this log for any errors related to KAAJEE and take appropriate actions to remedy the error.

Sign-On Log

This event log records all users that sign onto the VistA M Server via Kernel in the SIGN-ON LOG file (#3.081)**.** IRM should monitor this log. IRM should check for unusual activity (e.g., unusual amount of activity for a given user). If there is an unusual amount of activity for a particular user, IRM should further investigate by contacting the user in question and taking appropriate action as deemed appropriate.

Failed Access Attempts Log

This event log records users that fail to enter a valid Access/Verify code pair**.** IRM should monitor this log. IRM should check for unusual activity (e.g., unusual amount of activity for a given user). If there is an unusual amount of activity for a particular user, IRM should further investigate by contacting the user in question and taking appropriate action as deemed appropriate.

Remote Access/Transmissions

For every user logon, Web browser applications on the client workstation transmit/receive data using Hyper Text Transport Protocol (HTTP) to communicate with KAAJEE-enabled applications deployed on the application server.

NOTE: HTTP rides over Transmission Control Protocol/Internet Protocol (TCP/IP) in the payload packet.

On the application server, KAAJEE-enabled Web-based applications call the KAAJEE login/authentication component, which then calls VistALink using APIs. VistALink uses Transmission Control Protocol/Internet Protocol (TCP/IP) to transmit data to and receive data from VistA M Servers.

The KAAJEE SSPIs on the application server use Java Database Connector (JDBC) to query the remote security store database (e.g., Oracle), which holds the temporary username and password. KAAJEE also uses the SDS APIs to query tables on the remote national SDS database.

After authentication, applications can optionally make subsequent VistALink calls to run any RPCs authorized to the authenticated user.

Interfaces

The KAAJEE and KAAJEE SSPIs software interfaces with the following VA software:

- VistALink 1.5 (fully patched)
- Standard Data Services (SDS) tables 3.0 (or higher).

NOTE: KAAJEE works with SDS 3.0 or higher; however, KAAJEE 1.0.1.xxx distributes SDS 13.0 client jar files as part of the Sample Web Application. If you deploy the both the KAAJEE Sample Web Application and your own Web-based application on the same WebLogic Application Server domain instance and intend to use a different version of SDS, those client jar files will need to be swapped out for the appropriate version of the SDS client jar files. Otherwise, There may be a conflict if both applications reference the same JNDI tree.

REF: For more information on SDS tables, please visit the following Website:

http://vaww.sts.infoshare.va.gov/STS_SDS/Project%20Artifacts/Forms/AllItems.aspx

KAAJEE and KAAJEE SSPIs interfaces with the following *non*-VA Commercial-Off-The-Shelf (COTS) products/software:

- Oracle or Caché databases.
- WebLogic 8.1 (SP4 or higher) Application Server

NOTE: There are *no* other COTS (*non*-VA) products embedded in or requiring special interfaces by this version of KAAJEE, other than those provided by the underlying operating systems.

Electronic Signatures

There are *no* electronic signatures used within KAAJEE 1.0.1.xxx.

Security Keys

The following VistA M Server security key is exported with KAAJEE 1.0.1.xxx:

Table 9-1. KAAJEE exported security keys

Security Key	Description
XUKAAJEE_SAMPLE	This security key is exported with Kernel Patch XU*8.0*451. This security key is required in order to access the KAAJEE Sample Web Application exported with KAAJEE. First, an initial authentication occurs against a VistA M Server (i.e., Access and Verify codes). Then, if the login user passes this phase, the XUKAAJEE_SAMPLE VistA M security key is used to create a J2EE group/principal of the same name on the J2EE Application Server, if not already created. In addition, the login user will be assigned membership to this group on the J2EE Application Server during the login session. This membership is necessary as the authorization aspect of Form-based Authentication validates the role-based access by the membership of the associated group/principal.

File Security

There are *no* new file or field security changes associated with KAAJEE.

Contingency Planning

All sites should develop a local contingency plan to be used in the event of software/hardware problems in a production (live) environment. The contingency plan *must* identify the procedure for maintaining functionality provided by this software in the event of system outage.

Official Policies

There are *no* special legal requirements involved in the use of KAAJEE.

Distribution of KAAJEE is unrestricted.

As per the Software Engineering Process Group/Software Quality Assurance (SEPG/SQA) Standard Operating Procedure (SOP) 192-039—Interface Control Registration and Approval (effective 01/29/01), application programmers *must* not alter any Health***e***Vet VistA Class I software code.

REF: For more information on SOP 192-039—Interface Control Registration and Approval, please refer to the following Website:

http://vista.med.va.gov/SEPG_lib/Standard%20Operating%20Procedures/192-039%20Interface%20Control%20Registration%20and%20Approval.htm

10. Cactus Testing with KAAJEE

Cactus is a simple test framework for unit testing server-side Java code (servlets, Enterprise JavaBeans [EJBs], tag libs, filters, etc.).[15] Kernel Authentication and Authorization Java (2) Enterprise Edition (KAAJEE) supports testing with the Cactus container-based unit testing tool. It is possible that other container-based unit testing tools could be supported as well, but Cactus is the one that is the basis of developing KAAJEE's unit test support.

NOTE: This chapter assumes that the reader has basic understanding of the Jakarta Cactus unit testing tool.

REF: For more information on the Cactus testing tool, please visit the Jakarta Cactus Website at the following Website:

http://jakarta.apache.org/cactus/

Enabling Cactus Unit Test Support

To enable Cactus unit test support, do the following:

1. Switch from FORM to BASIC authentication. For example, In your J2EE Web-based application's web.xml, code as follows:

Figure 10-1. Switching from FORM to BASIC in web.xml example

```
<login-config>
<auth-method>BASIC</auth-method>
<!-- <auth-method>FORM</auth-method>
 <form-login-config>
   <form-login-page>/login/login.jsp</form-login-page>
   <form-error-page>login/loginerror.jsp</form-error-page>
 </form-login-config> -->
</login-config>
```

2. Turn on Cactus Support in the KAAJEE configuration file, set the following tag to "true" (case sensitive):

```
<cactus-insecure-mode enabled="true" />
```

This mode should *never* be enabled on a production system. It defaults to "false" unless enable is specifically set to "true" (case sensitive).

Essentially, this switch turns the "one-time login token" to a "many-time login token," allowing the re-use of login credentials over repeated Cactus unit tests.

---

[15] "The Apache Jakarta Project;" Overview of Cactus Website: http://jakarta.apache.org/cactus/, last updated 2/15/05.

3. Add the normal required Cactus configuration information into your application's web.xml.

Using Cactus in a KAAJEE-Secured Application

There are probably several approaches to obtaining a login credential on your Cactus test client side, to use to login on the container side. Essentially:

- Start with a valid-for-login Access code, Verify code and Division.
- Pass these, on the container side, to the LoginController.getFormsAuthCredentialsForCactus().
- The return value (for valid login credentials) is an object that contains valid j_username and j_password values.

How do you do this?

One approach is:

1. Configure both secured and unsecured Cactus test redirector servlets in your Web-based application's web.xml deployment descriptor.
2. Create one Cactus test in your test suite that uses an unsecured ServletRedirector Cactus test redirector servlet. This application will gather a set of login credentials from the server.
    - The beginXXX, testXXX and endXXX methods should be, sequentially in your test class source code, the first set of tests. Cactus/JUnit appear to follow source code order when sequencing test execution.
    - This unsecured Cactus test should start with the Access code, Verify code, and Division. These could be hard-coded into the test class, or could be kept in a client-side configuration file, read on the client during the beginXXX method, and passed to the server-side testXXX method as session attributes.
    - In the server-side testXXX method, call the KAAJEE LoginController class's static getFormsAuthCredentialsForCactus method to obtain a valid j_username and j_password value. These are returned in a KAAJEE CactusFormsAuthCredentialVO object.
    - In the server-side testXXX method, you could also obtain and cache the LoginUserInfoVO object. The getFormsAuthCredentialsForCactus will put this into the testXXX method's session object if you pass that as a parameter. You need to store this somewhere on the server so you can retrieve it in subsequent testXXX methods; the example below stores it in a static class variable in the server-side version of the test class.
    - Using the toString() method of the CactusFormsAuthCredentialVO object, write the credentials to the Web page output, using the servlet's PrintWriter.
    - Back on the client in the endXXX method, instantiate a new CactusFormsAuthCredentialVO object, using the complete Web response as input. The CactusFormsAuthCredentialVO class can instantiate itself by looking for its own "toString" output in any given string.

- On the client side, again in the endXXX method, store these values (the CactusFormsAuthCredentialVO object provides a valid j_username and j_password) in a static class variable in the test class.

3. If you are caching the LoginUserInfoVO object in the server instance of the test class, you could add code in the setUp() method (executed before every testXXX method) to put the LoginUserInfoVO object back into session, for use as needed by each testXXX method.
4. For the rest of the Cactus tests in the test class, use the secured ServletRedirector (specify in the beginXXX method), and pass the credentials using a Cactus BasicAuthentication object. The testXXX methods should all run in the server context created by the login of the secure ServletRedirector.

This approach has been tested for ServletTestCase. While it should work for JspTestCase, it has not been tested.

Cactus ServletTestCase Example

Figure 10-2. Cactus ServletTestCase example

```
package gov.va.med.authentication.kernel.samples;

import java.io.IOException;
import java.io.PrintWriter;

import gov.va.med.authentication.kernel.CactusFormsAuthCredentialVO;
import gov.va.med.authentication.kernel.KaajeeLoginException;
import gov.va.med.authentication.kernel.LoginController;
import gov.va.med.authentication.kernel.LoginUserInfoVO;
import junit.framework.Test;
import junit.framework.TestSuite;

import org.apache.cactus.ServletTestCase;
import org.apache.cactus.WebRequest;
import org.apache.cactus.WebResponse;
import org.apache.cactus.client.authentication.BasicAuthentication;

public class TestSampleApp extends ServletTestCase {

     /**
      * Access code, Verify code, Division to transform
      * into a valid j_username, j_password
      */
  private static final String userAccessCode = "good!@#$1";
  private static final String userVerifyCode = "good!@#$2";
  private static final String userDivision = "631";

     /**
      * To store the login credentials on the client side
      */
  private static CactusFormsAuthCredentialVO clientCredentials;
     /**
      * To store the userInfo session object on the server side
      * (problematic if multiple tests are running simultaneously)
      */
  private static LoginUserInfoVO serverUserInfo;

  public TestSampleApp(String theName) {
    super(theName);
  }

  public static Test suite() {

    TestSuite testSuite = new TestSuite(TestSampleApp.class);
    return testSuite;
  }

  /**
   * Runs on SERVER, before every test.
   */
  public void setUp() {
    // put login user info object back into session, if
    // already cached in the static class variable.
    if (serverUserInfo!= null) {
      session.setAttribute(LoginUserInfoVO.SESSION_KEY, serverUserInfo);
    }
  }
```

```
/**
 * Runs on SERVER, after every test.
 */
public void tearDown() {
}

/**
 * Runs on CLIENT. Communicates with server-side test execution via
 * WebRequest. This test should use a _unsecured_ Cactus redirector.
 */
public void beginGetServerCredentials(WebRequest webRequest) {
     // use unsecured test director (depends on web.xml configuration)
  webRequest.setRedirectorName("ServletRedirector");
}

/**
 * Runs on SERVER. Gets the j_username and j_password login
 * credentials, and returns to client by writing to the servlet output.
 */
public void testGetServerCredentials() {
  try {
    PrintWriter out = response.getWriter();
       // call KAAJEE to get valid j_username/j_password credentials
       // based on a/v code, division
       CactusFormsAuthCredentialVO serverCredentials =
         LoginController.getFormsAuthCredentialsForCactus(
           userDivision,
           userAccessCode,
           userVerifyCode,
        session);
    // return credentials to client via servlet output
    out.println("testGetServerCredentials credentials: " +
        serverCredentials.toString());
    // get LoginUserInfoVO from session; save in static
    // class variable in server side test class (not OK
    // if multiple tests are running simultaneously)
    serverUserInfo = (LoginUserInfoVO)
        session.getAttribute(LoginUserInfoVO.SESSION_KEY);
  } catch (KaajeeLoginException e) {
    //
  } catch (IOException e) {
    //
  }
}

/**
 * Runs on CLIENT. Stores login credentials returned from server
 * in static class variable (client-side).
 */
public void endGetServerCredentials(WebResponse webResponse) {
     System.out.println(webResponse.getText());
  // parse, store login credentials returned from server
  clientCredentials = new
    CactusFormsAuthCredentialVO(webResponse.getText());
}

/**
 * Runs on CLIENT. This test should use a _secure_ Cactus redirector,
 * using credentials gathered by earlier test run.
 */
public void beginSecureLogin(WebRequest webRequest) {
  // use secured test director (depends on web.xml configuration)
```

```
    webRequest.setRedirectorName("ServletRedirectorSecure");
    // use credentials obtained previously to login
    webRequest.setAuthentication(
      new BasicAuthentication(clientCredentials.getJUsername(),
      clientCredentials.getJPassword()));
  }

  /**
   * Runs on SERVER.
   */
  public void testSecureLogin() {
    // test should be running in the KAAJEE-created container
    // security context of the KAAJEE-logged-in ServletRedirectSecure.
    //
    // Now run any tests you need, in the KAAJEE
    try {
      PrintWriter out = response.getWriter();
         // display security context info
         out.println("getRemoteUser: " + request.getRemoteUser());
    } catch (IOException e) {
      //
    }
  }

  /**
   * Runs on CLIENT.
   */
  public void endSecureLogin(WebResponse webResponse) {
        System.out.println(webResponse.getText());
  }
}
```

Other Approaches *Not* Recommended

It would be possible to insert a valid j_username and j_password directly into the kaajeeweblogontoken table. Reasons not to do this include:

- The LoginUserInfoVO object will not be created.
- The proper DUZ for the given Access and Verify code is guaranteed when obtained from the LoginUserInfoVO object.
- Going through the full process of translating an Access/Verify code at runtime into a login credential assures that there are no problems (login-wise) with the M account being connected to.
- The tables are purged at every server restart, destroying the credential.
- Inserting malformed credentials into the table may cause login problems.

Another approach is to use the LoginController's getFormsAuthCredentialsForCactus method to get a valid credential once, store this credential on the client, and re-use between tests. This approach has the most of the same drawbacks as the first alternate method described above.

11. Troubleshooting

Common Login-related Error Messages

This chapter describes some of the common Kernel Authentication and Authorization Java (2) Enterprise Edition (KAAJEE) and VistALink-related error messages that users might encounter during the Authentication and Authorization process of KAAJEE-enabled applications. For each error message listed, we include the cause and suggest possible resolutions to correct the error. All KAAJEE/VistALink error messages are displayed in an HTML format (i.e., Web page) in any of the following template files:

- loginerror.jsp
- loginerrordisplay.jsp
- navigatonerrordisplay.jsp

These files are located in the following directory:

<STAGING_FOLDER>\kaajee-1.0.1.xxx\jars\jsp\login\

The following error messages are discussed in this chapter:

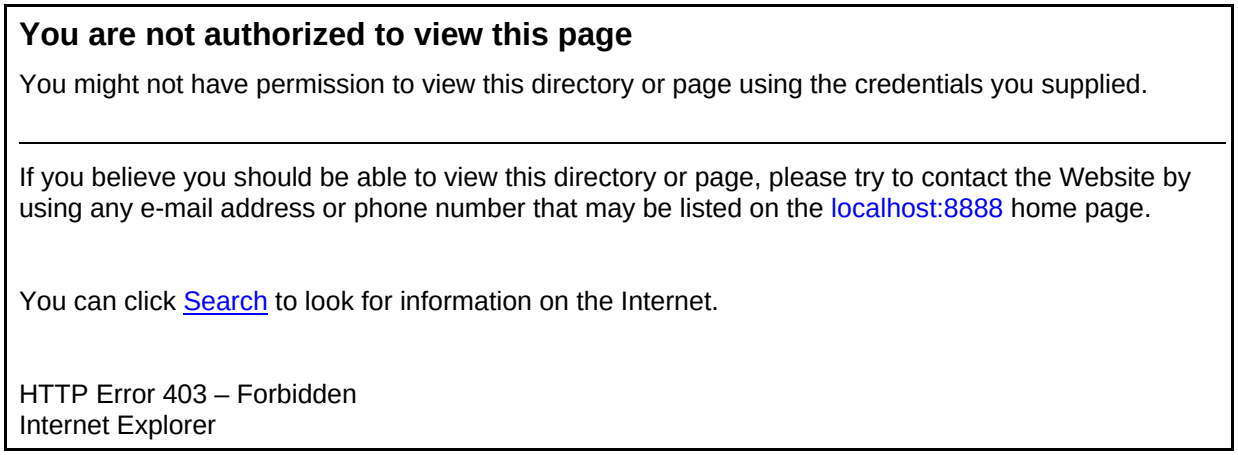
- **Error: You are not authorized to view this page**
- **Error: Forms authentication login failed**
- **Error: You navigated inappropriately to this page**
- **Error: Could not get a connection from connector pool**
- **Error: Error retrieving user information**
- **Error: Authorization failed for your user account on the M system**
- **Error: Login failed due to too many invalid logon attempts**
- **Error: Your verify code has expired or needs changing**
- **Error: Not a valid ACCESS CODE/VERIFY CODE pair**
- **Error: Logins are disabled on the M system**
- **Error: Could not match you with your M account**
- **Error: Institution/division you selected for login is not valid for your M user account**
- **Error: Error logging on or retrieving user information**

NOTE: The error messages discussed in this chapter are *not* listed in any particular order.

Error: You are not authorized to view this page

Message:

Figure 11-1. Error—You are not authorized to view this page



Cause: The user attempts to access a protected resource, and instead of being prompted for their login credentials, they are immediately given a Hyper Text Transport Protocol (HTTP) Error 403 (not authorized) error (Figure 11-1).

Some possible reasons that the authorization may have failed:

- Lack of Proper Security Keys—The end-user's account does not have the VistA M Server J2EE security keys matching the role required for this page.
- Error Retrieving User Roles—Some other error prevented proper retrieval of user roles during the login process.

Resolution: For the following situations, the user *must* contact IRM or the System Administrator for assistance:

- Lack of Proper Security Keys—Get the necessary VistA M Server J2EE security keys assigned.
- Error Retrieving User Roles—Check the log4J logs for any errors.

Error: Forms authentication login failed

Message:

Figure 11-2. Error—Forms authentication login failed

Cause: The user enters their Access and Verify codes and presses the **Login** button. No obvious error is returned, but the user is sent to the loginerror.jsp error page (error message template) that states a generic message: "Forms authentication login failed." (Figure 11-2).

The user was redirected by J2EE Form-based Authentication to the login error page (as specified in the <form-error-page> tag of the <login-config> tag of the web.xml file.

Some possible reasons that the authentication may have failed:

- Lack of Proper Security Keys—The user does not possess the VistA M Server J2EE Security Key corresponding to the WebLogic group name (corresponding to the J2EE security role) required to access the protected resource. In such cases, the user will be sent to the login error page stating that "Forms authentication login failed".
- WebLogic Configuration Problem—The WebLogic Custom Security Authentication Providers are not configured correctly.

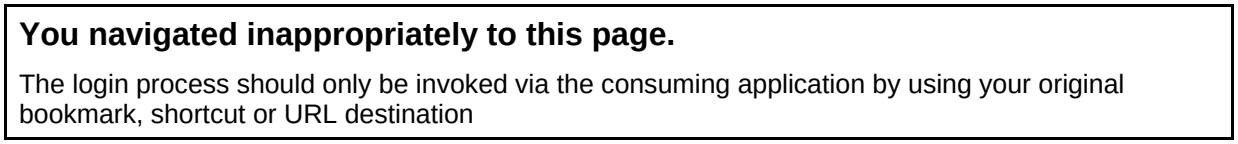
Resolution: For the following situations, the user *must* contact IRM or the System Administrator for assistance:

- Lack of Proper Security Keys—If Log4J is configured to log the gov.va.med.authentication.kernel package for DEBUG level messages, examine the Log4J log files for output from the class UserManagerImp. If you see a login attempt from the user in question and a return value from the UserManagerImp.inGroupName method of false, then the user does not have the necessary authorizations.
- WebLogic Configuration Problem—If you have Log4J configured to log the gov.va.med.authentication.kernel package for DEBUG level messages, examine the Log4J log files for output from the class UserManagerImp. If no such output is present, the WebLogic Custom Security Authentication Providers are probably *not* configured correctly in weblogic.xml, or the application did not deploy correctly.

Error: You navigated inappropriately to this page

Message:

Figure 11-3. Error—You navigated inappropriately to this page



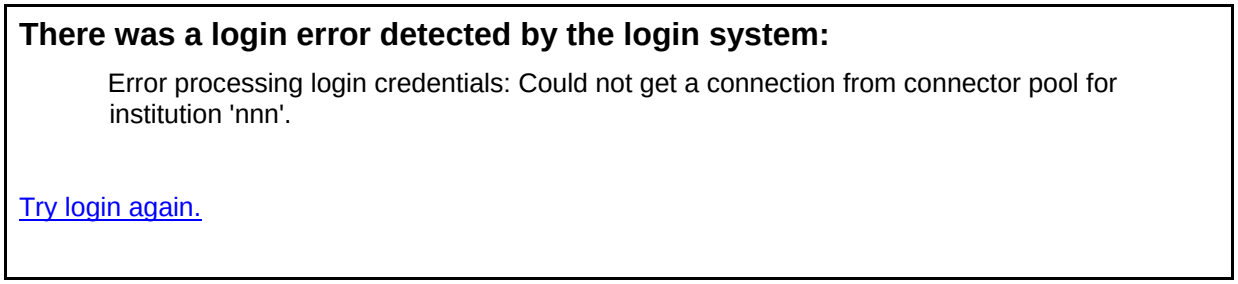
Cause: After successfully logging into the Web application, the user presses the browser **Back** button until they reach the level of the KAAJEE Web login page. This message is displayed by the navigatonerrordisplay.jsp.

Resolution: Because the user is already successfully logged into the Web application, they should just press the browser **Forward** button to get back to the desired Web application page.

Error: Could not get a connection from connector pool

Message:

Figure 11-4. Error—Could not get a connection from connector pool



Cause: The user enters their Access and Verify codes and presses the **Login** button. The user is then redirected to the loginerrordisplay.jsp error page (error message template) with a descriptive error message displayed (Figure 11-4).

In this case, the descriptive error message stated that the system could not get a connection from the connector pool for the institution selected by the user.

Several possible reasons for this failure include:

- No Institution mapping is configured to associate Station Number **nnn** (e.g., 662) with a JNDI name of a connector.
- No connector exists for the mapped JNDI name returned by VistALink's Institution Mapping.
- The VistA M Server to which the connector is connecting is down.

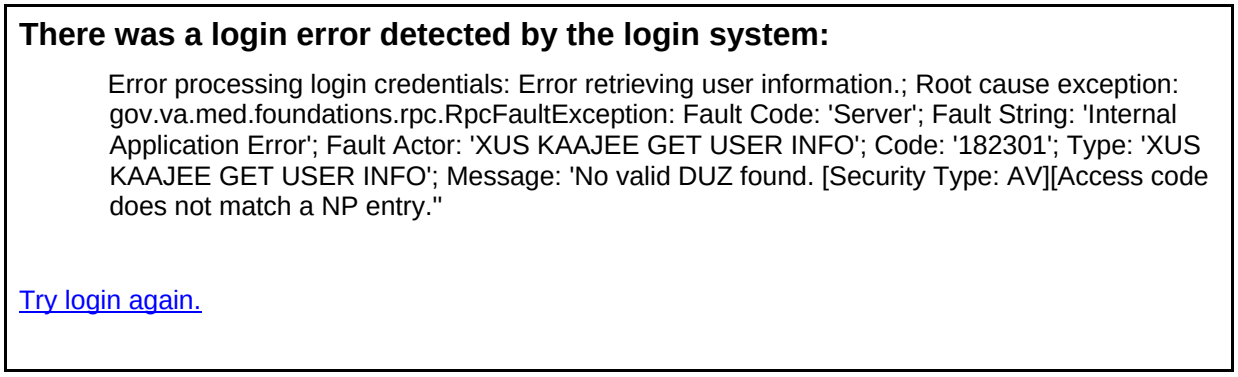
Resolution: The user *must* contact IRM or the Systems Administrator for assistance. A review of the

log files for both the application and the connector should further narrow down the exact cause of the failure.

Error: Error retrieving user information

Message:

Figure 11-5. Error—Error retrieving user information



Cause: The user enters their Access and Verify codes and presses the **Login** button. The user is then redirected to the loginerrordisplay.jsp error page (error message template) with a descriptive error message displayed (Figure 11-5).

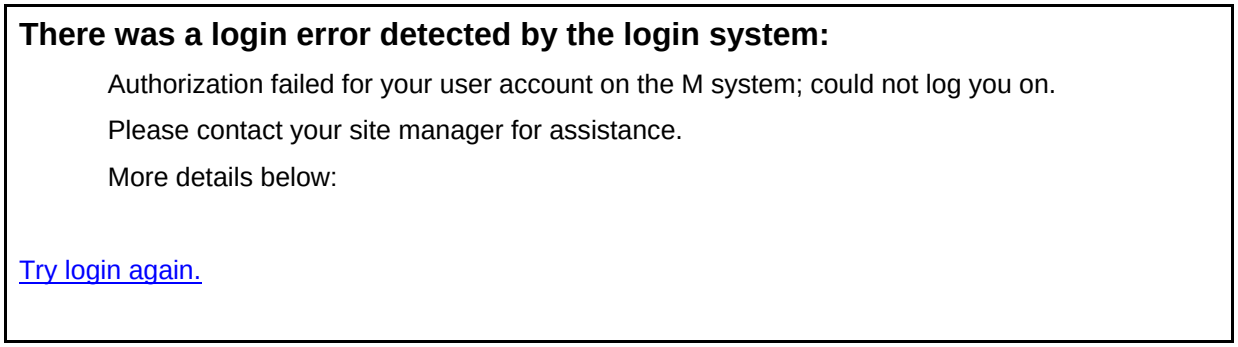
In this case, the descriptive error message stated that the system could not find a valid DUZ for the user. The Access code entered by the user was not found in the NEW PERSON file (#200).

Resolution: The user *must* contact IRM or the Systems Administrator to verify that the user is allowed access to the VistA M Server account in question and then grant the user appropriate access.

Error: Authorization failed for your user account on the M system

Message:

Figure 11-6. Error—Authorization failed for your user account on the M system



Cause: The user enters their Access and Verify codes and presses the **Login** button. The user is then redirected to the loginerrordisplay.jsp error page (error message template) with a descriptive error message displayed (Figure 11-6).

Several possible reasons for this failure include:

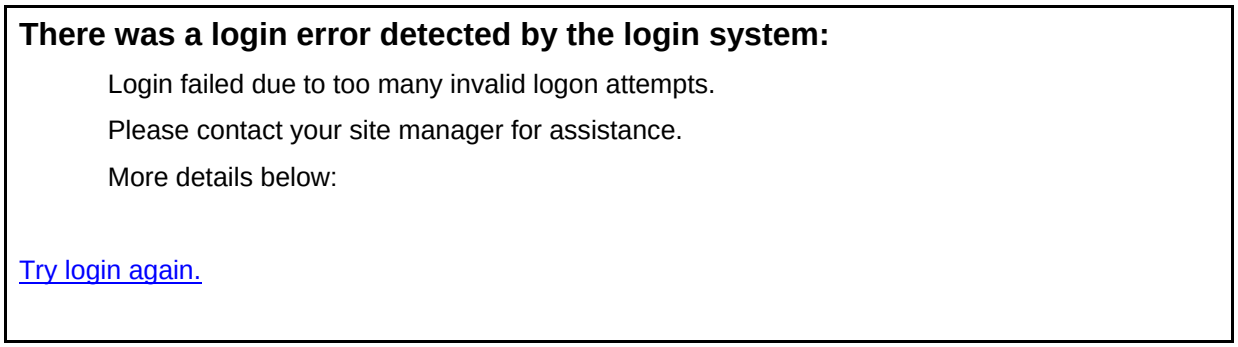
- The user is not authorized to access the VistA M Server in question.
- The user is not set up correctly on the VistA M Server in question.

Resolution: The user *must* contact IRM or the Systems Administrator to verify that the user is allowed access to the VistA M Server account in question and then grant the user appropriate access.

Error: Login failed due to too many invalid logon attempts

Message:

Figure 11-7. Error—Login failed due to too many invalid logon attempts



Cause: The user enters their Access and Verify codes and presses the **Login** button. The user is then redirected to the loginerrordisplay.jsp error page (error message template) with a descriptive error message displayed (Figure 11-7).

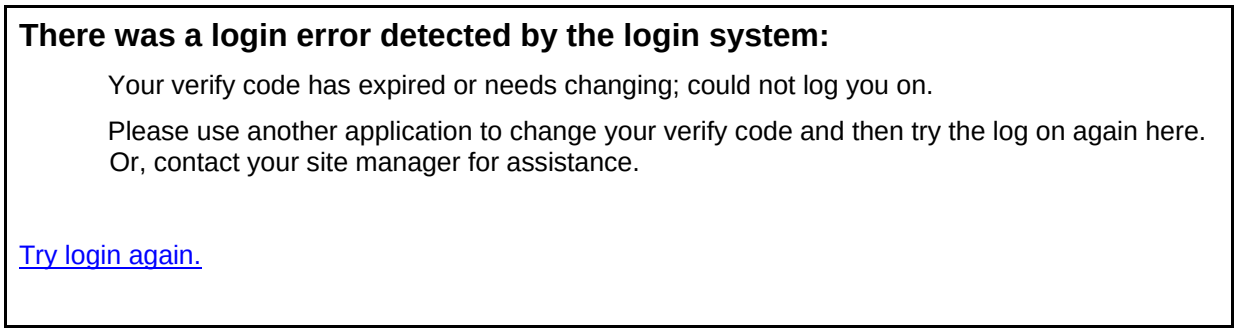
The user has exceeded the allowed number of login attempts to the VistA M Server and *must* wait a prescribed period of time before attempting another login.

Resolution: If after the prescribed wait period has passed and the user tries to log back into the VistA M Server, and again fails in the attempt, the user *must* contact IRM or the System Administrator for assistance.

Error: Your verify code has expired or needs changing

Message:

Figure 11-8. Error—Your verify code has expired or needs changing



Cause: The user enters their Access and Verify codes and presses the **Login** button. The user is then redirected to the loginerrordisplay.jsp error page (error message template) with a descriptive error message displayed (Figure 11-8).

Several possible reasons for this failure include:

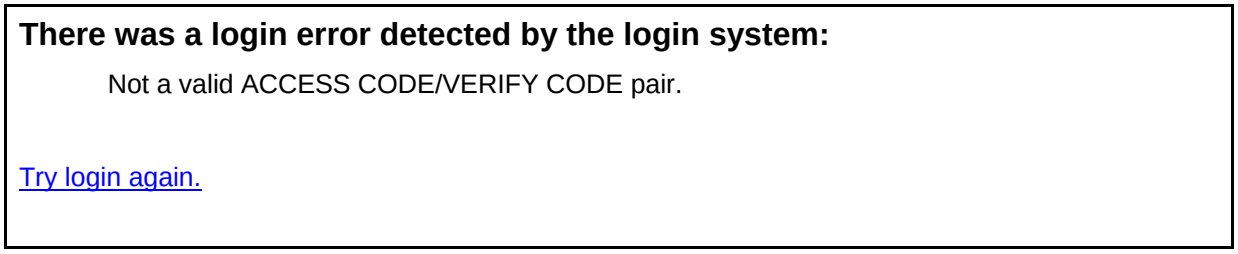
- The user's Verify code has expired a predefine time limit and *must* be changed before being allowed to access the VistA M Server.
- The user is given a temporary Verify code because they are new to the VistA M Server or asked IRM to give them new access. Upon their first login, this temporary Verify code expires immediately and *must* be changed.

Resolution: Since KAAJEE-enabled Web-based applications do *not* support changing your Verify code at this time, users *must* use another *non*-KAAJEE-enabled Web-based application in order to be prompted to change their Verify code.

Error: Not a valid ACCESS CODE/VERIFY CODE pair

Message:

Figure 11-9. Error—Not a valid ACCESS CODE/VERIFY CODE pair



Cause: The user enters their Access and Verify codes and presses the **Login** button. The user is then redirected to the loginerrordisplay.jsp error page (error message template) with a descriptive error message displayed (Figure 11-9).

Several possible reasons for this failure include:

- The user has entered an incorrect Access code.
- The user has entered an incorrect Verify code.
- The user has entered both an incorrect Access *and* Verify code.
- The user is not allowed access to the VistA M Server in question.
- The user was not set up correctly on the VistA M Server in question.

For security reasons, the system does *not* specify which code was entered incorrectly.

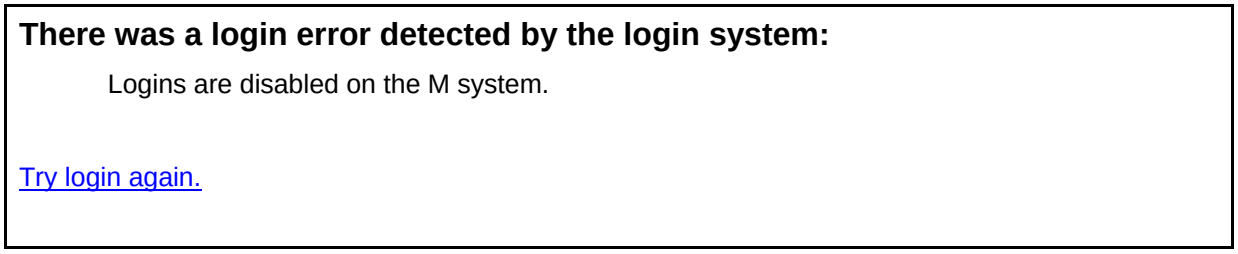
Resolution: The user should re-enter the correct Access and Verify codes.

If the error persists, the user *must* contact IRM or the System Administrator to verify that the user is allowed access to the VistA M Server account in question and then grant the user appropriate access.

Error: Logins are disabled on the M system

Message:

Figure 11-10. Error—Logins are disabled on the M system



Cause: The user enters their Access and Verify codes and presses the **Login** button. The user is then redirected to the loginerrordisplay.jsp error page (error message template) with a descriptive error message displayed (Figure 11-10).

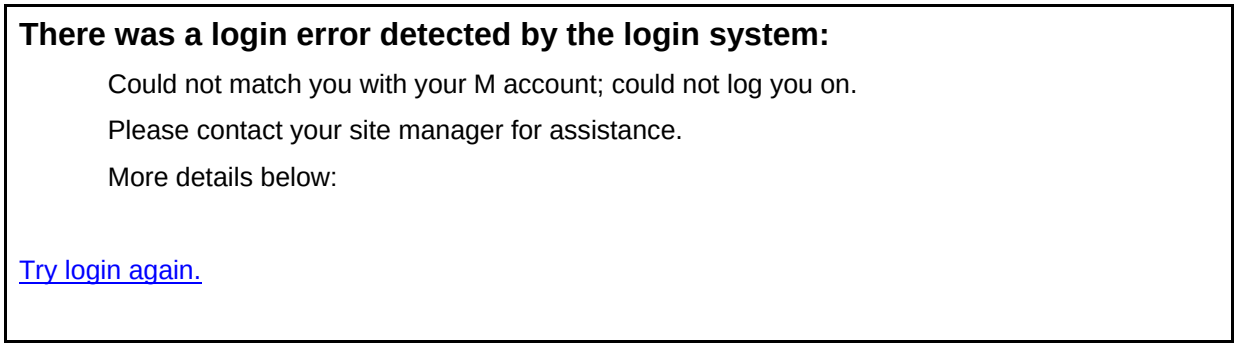
IRM or the System Administrator has disabled logins on the VistA M Server. Logins are sometimes disabled in order to install new software or perform system maintenance.

Resolution: The user should wait and try to log into the VistA M Server at a later time. If the user feels the time period to log back into the system is excessive, the user should contact IRM or the System Administrator for assistance.

Error: Could not match you with your M account

Message:

Figure 11-11. Error—Could not match you with your M account



Cause: The user enters their Access and Verify codes and presses the **Login** button. The user is then redirected to the loginerrordisplay.jsp error page (error message template) with a descriptive error message displayed (Figure 11-11).

Several possible reasons for this failure include:

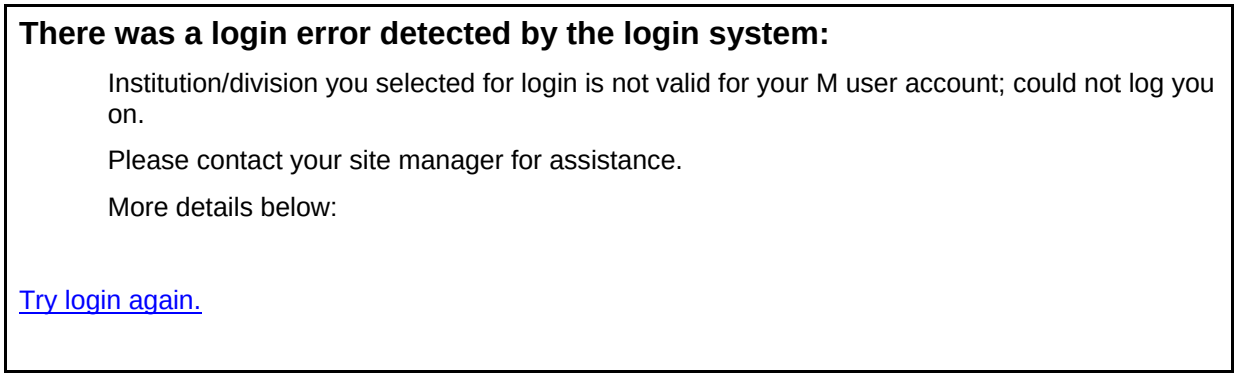
- The user is not allowed access to the VistA M Server in question.
- The user was not set up correctly on the VistA M Server in question.
- The user has entered an incorrect Access code.
- The user has entered an incorrect Verify code.
- The user has entered both an incorrect Access *and* Verify code.

Resolution: The user *must* contact IRM or the System Administrator to verify that the user is allowed access to the VistA M Server account in question and then grant the user appropriate access.

Error: Institution/division you selected for login is not valid for your M user account

Message:

Figure 11-12. Error—Institution/division you selected for login is not valid for your M user account



Cause: The user enters their Access and Verify codes and presses the **Login** button. The user is then redirected to the loginerrordisplay.jsp error page (error message template) with a descriptive error message displayed (Figure 11-12).

Several possible reasons for this failure include:

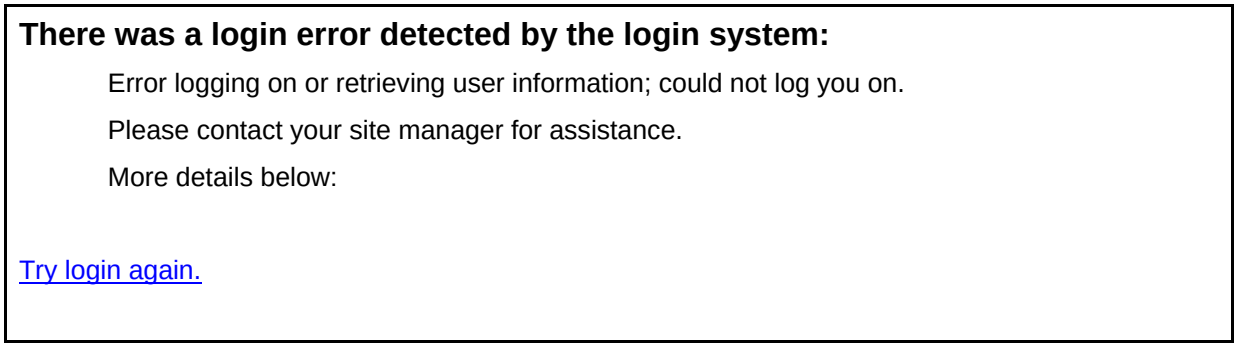
- The user does not have the selected Institution/Division entry in the DIVISION Multiple field (#16) in the NEW PERSON file (#200) entry.
- The SDS tables could *not* validate the Division selected.

Resolution: The user *must* contact IRM or the System Administrator to verify that the user is allowed access to the Institution/Division in question and then grant the user appropriate access.

Error: Error logging on or retrieving user information

Message:

Figure 11-13. Error—Institution/division you selected for login is not valid for your M user account



Cause: The user enters their Access and Verify codes and presses the **Login** button. The user is then redirected to the loginerrordisplay.jsp error page (error message template) with a descriptive error message displayed (Figure 11-11).

Several possible reasons for this failure include:

- The user is not allowed access to the VistA M Server in question.
- The user was not set up correctly on the VistA M Server in question.
- The user has entered an incorrect Access code.
- The user has entered an incorrect Verify code.
- The user has entered both an incorrect Access *and* Verify code.

Resolution: The user *must* contact IRM or the System Administrator to verify that the user is allowed access to the VistA M Server account in question and then grant the user appropriate access.

REF: For a list of other login-related error messages, please refer to the "Symptoms and Possible Solutions" topic in Chapter 7 in the VistALink *System Administration Guide*.

REF: For more information on the Kernel signon process and related error messages, please refer to the "Signon/Security" section in the *Kernel Systems Management Guide*.

12. Appendix A—Sample Deployment Descriptors

All KAAJEE sample deployment descriptors are located in the following KAAJEE directory (i.e., kaajee-1.0.1.xxx):

<STAGING_FOLDER>\kaajee-1.0.1.xxx\dd_examples

REF: For a sample of the kaajeeConfig.xml file, please refer to Figure 6-2 in chapter 6, "KAAJEE Configuration File," in this manual.

application.xml

Figure 12-1. Sample KAAJEE Deployment Descriptor: application.xml file (e.g., KAAJEE sample application)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application
1.3//EN" "http://java.sun.com/dtd/application_1_3.dtd">
<application>
    <display-name>KaajeeSampleEar</display-name>
    <module>
        <web>
            <web-uri>kaajeeSampleApp.war</web-uri>
            <context-root>/kaajeeSampleApp</context-root>
        </web>
    </module>
</application>
```

Application developers would customize this sample descriptor for their use by replacing the following information with information specific to their application:

- **<display-name> Tag—**Replace "KaajeeSampleEar" ear file name with the name of your application ear file.
- **<web-uri> Tag—**Replace "kaajeeSampleApp.war" war file name with the name of your application war file.
- **<context-root> Tag—**Replace "/kaajeeSampleApp" root directory with the name of your application root directory.

web.xml

Figure 12-2. Sample KAAJEE Deployment Descriptor: web.xml file (e.g., KAAJEE Sample Web Application)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>

  <listener>
    <listener-class>
      gov.va.med.authentication.kernel.KaajeeSessionAttributeListener
    </listener-class>
  </listener>

  <listener>
    <listener-class>
      gov.va.med.authentication.kernel.KaajeeHttpSessionListener
    </listener-class>
  </listener>

  <servlet>
    <servlet-name>SampleAppInit</servlet-name>
    <servlet-
class>gov.va.med.authentication.kernel.samples.InitSampleAppServlet</servlet-class>
    <init-param>
      <param-name>log4j-init-file</param-name>
      <param-value>/log4jConfig.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

   <servlet>
    <servlet-name>KaajeeInit</servlet-name>
    <servlet-class>gov.va.med.authentication.kernel.InitKaajeeServlet</servlet-
class>
    <init-param>
      <param-name>kaajee-config-file-location</param-name>
      <param-value>/WEB-INF/kaajeeConfig.xml</param-value>
    </init-param>
    <load-on-startup>3</load-on-startup>
  </servlet>

  <servlet>
    <servlet-name>LoginController</servlet-name>
    <servlet-class>gov.va.med.authentication.kernel.LoginController</servlet-class>
  <run-as>
  <!-- In this example, weblogic is the boot user name (i.e., weblogic console user
name)  -->
<role-name>weblogic</role-name>
</run-as>
  </servlet>

  <servlet-mapping>
    <servlet-name>LoginController</servlet-name>
    <url-pattern>/LoginController</url-pattern>
  </servlet-mapping>

  <session-config>
    <session-timeout>2</session-timeout>
  </session-config>
```

```
  <error-page>
    <error-code>404</error-code>
    <location>/AppErrorPage.jsp</location>
  </error-page>

   <security-constraint>
    <web-resource-collection>
      <web-resource-name>A Protected Page</web-resource-name>
      <url-pattern>/AppHelloWorld.jsp</url-pattern>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
      </web-resource-collection>
      <auth-constraint>
        <role-name>XUKAAJEE_SAMPLE_ROLE</role-name>
      </auth-constraint>
      <user-data-constraint>
        <transport-guarantee>NONE</transport-guarantee>
      </user-data-constraint>
  </security-constraint>

   <login-config>
   <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>login/login.jsp</form-login-page>
      <form-error-page>login/loginerror.jsp</form-error-page>
    </form-login-config>
   </login-config>

  <security-role>
    <description>KERNEL KAAJEE Sample role</description>
    <role-name>XUKAAJEE_SAMPLE_ROLE</role-name>
  </security-role>

  <security-role>
     <role-name>AUTHENTICATED_KAAJEE_USER</role-name>
  </security-role>

</web-app>
```

Application developers would customize this sample descriptor for their use by adding in their application servlets and by replacing the following information with information specific to their application:

- **<security-constraint> Tag (multiple):**
  - **<url-pattern> Tag—**Replace "**/AppHelloWorld.jsp**" security constraint URL with your application's security constraint URL.
  - **<role-name> Tag—**Replace "**XUKAAJEE_SAMPLE_ROLE**" security constraint role name with your application's security constraint role name.
- **<security-role> Tag (multiple):**
  - **<description> Tag—**Replace/add all security role descriptions (e.g., "**KERNEL KAAJEE Sample role**") with your application's security role descriptions.
  - **<role-name> Tag—**Replace/add all security role names (e.g., "**XUKAAJEE_SAMPLE_ROLE**") with your application's security role names.

weblogic.xml

The BEA weblogic.xml file is used to map security role names (i.e., <security-role> element entries in the web.xml file) to users and/or groups (i.e., principals); KAAJEE only uses groups. The WebLogic Application Server will only allow mapped security roles access to protected URL resources.

REF: For a sample spreadsheet showing a mapping between WebLogic group names (i.e., principals) with J2EE security role names, please refer to "Appendix B—Mapping WebLogic Group Names with J2EE Security Role Names" in this manual.

Figure 12-3. Sample KAAJEE Deployment Descriptor: weblogic.xml file (e.g., KAAJEE Sample Web Application)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web
Application 7.0//EN"
"http://www.bea.com/servers/wls700/dtd/weblogic700-web-jar.dtd">
<weblogic-web-app>
<security-role-assignment>
<role-name>XUKAAJEE_SAMPLE_ROLE</role-name>
<principal-name>XUKAAJEE_SAMPLE</principal-name>
</security-role-assignment>

  <session-descriptor>
    <session-param>
      <param-name>CookieName</param-name>
      <param-value>kaajeeJSESSIONID</param-value>
    </session-param>

  </session-descriptor>

</weblogic-web-app>
```

Application developers would customize this sample descriptor for their use by replacing the following information with information specific to their application:

- **<security-role-assignment> Tag:**
  - **<role-name> Tag—**Replace "**XUKAAJEE_SAMPLE_ROLE**" security role assignment role name with your application's security role assignment role name.
  - **<principal-name> Tag—**Replace "**XUKAAJEE_SAMPLE**" security role assignment principal name with your application's security role assignment principal name.
- **<session-param> Tag:**
  - **<param-value> Tag—**Replace "**kaajeeJSESSIONID**" security param value with your application's param value.

NOTE: Creating the weblogic.xml deployment descriptor is optional. If you do not include this file, or include the file but do not include mappings for all security roles, all security roles without mappings will default to any user or group whose name matches the role name.[16]

[16] Excerpt taken from the WebLogic Server™ Programming WebLogic Security Guide, Page 2-12; downloaded from the BEA Website: www.bea.com.

13. Appendix B—Mapping WebLogic Group Names with J2EE Security Role Names

The following table supersedes the role_mapping_worksheet.xls as delivered with KAAJEE 1.0.1.xxx. The role_mapping_worksheet.xls Microsoft Excel spreadsheet is located in the following directory:

<STAGING_FOLDER>\kaajee-1.0.1.xxx\dd_examples

Figure 13-1. Sample spreadsheet showing a mapping between WebLogic group names and J2EE security role names

VistA Security Key Name	**WebLogic Group Name** *(via WebLogic Console)*	**<security-role-assignment> subelement <principal-name> (i.e., group name)** **From: WebLogic group name...** *(weblogic.xml)*	**<security-role-assignment> subelement <role-name>** **...To: J2EE security role name** *(weblogic.xml)*	**J2EE <security-role> role-name** *(web.xml, ejb-jar.xml, application.xml)*
<---------- (WebLogic Group Names [a.k.a. Principals]) ---------->			**<---------- (J2EE Security Role Names) ---------->**	
DG-CLERK	DG-CLERK	DG-CLERK	CLERK	CLERK
DG-SUPERVISOR	DG-SUPERVISOR	DG-SUPERVISOR	SUPER	SUPER
DG-ADMIN	DG-ADMIN	DG-ADMIN	ADMIN	ADMIN

NOTE: The <security-role-assignment> elements in the weblogic.xml file are not needed when the <role-name> element and the <principal-name> element are the same. By default, WebLogic automatically creates a group of the same name if no mapping is defined in weblogic.xml.

Glossary

AA	Authentication and Authorization
DIVISION	Division is an Institution in the INSTITUTION file (#4) that is identified via a unique Station Number. Divisions are "sub"-divisions or child sites within an integrated set of facilities, whose computing is hosted on the computer system of the primary facility. The parent-child relationship between a division and a primary facility is maintained by the ASSOCIATIONS multiple field (#14) in the INSTITUTION file (#4). A sub-division may be a medical center, clinic, or nursing home. The primary facility is also a division of itself. Clinics and nursing homes are often sub-divisions. The Station Number for child sites is 5 characters, the first 3 of which are the 3 numbers of the parent facility. For example, Livermore, CA is a medical center that is a child of Palo Alto, CA. Its Station Number is 640A4.
EAR (file)	**E**nterprise **AR**chive file (.ear extension). This file has the same format as a regular .jar file. An ear file is like a zip file packaged for J2EE application deployment. The .ear file contains everything necessary to deploy an enterprise application on an application server. An ear file can contain 1-n Web modules. It contains at least one .war (Web Archive) file which contains the Web component of the application as well as the .jar (Java Archive) file. In addition, there are some deployment descriptor files in XML.[17]
EJB	**E**nterprise **J**ava**B**eans. Enterprise JavaBeans (EJB) technology is the server-side component architecture for the Java 2 Platform, Enterprise Edition (J2EE) platform. EJB technology enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology.[18]
HEALTHEVET-VISTA	The Health***e***Vet-VistA architecture is a services-based architecture. Applications are constructed in tiers with distinct user interface, middle, and data tiers. Two types of services will exist: Core Services—Infrastructure and data. Application Services—A single logical authoritative source of data.
HTTP SESSION OBJECT	Hyper Text Transport Protocol (HTTP) Session Objects are used like cookies to maintain states as Web pages are considered stateless rather than stateful.

[17] Derived from a question (What's an .ear file) posed by John Zukowski and defined by Mobushir Hingorjo on 3/6/00 (modified 8/4/00) and available on the following Web page: http://www.jguru.com/faq/view.jsp?EID=21097.
[18] Definition from the following Sun Microsystems Website: http://java.sun.com/products/ejb/.

J2EE	The Java 2 Platform, Enterprise Edition (J2EE) is an environment for developing and deploying enterprise applications. The J2EE platform consists of a set of services, APIs, and protocols that provide the functionality for developing multi-tiered, Web-based applications.
JAR (file)	**J**ava **AR**chive file (.jar extension). A collection of Java files compressed using the ZIP/ZLIB compression format.
JAVA	Java is a programming language. It can be used to complete applications that may run on a single computer or be distributed among servers and clients in a network.
JAVABEANS	JavaBeans expose methods, properties, and events, which can then be accessed by other components or scripts. JavaBeans are commonly mistaken for design patterns as they both use similar conventions (e.g., both use Setter and Getter methods). A JavaBean is a reusable component that can be used in any Java application development environment. JavaBeans are dropped into an application container, such as a form, and can perform functions ranging from a simple animation to complex calculations.[19]
JDBC	**J**ava **D**ata**b**ase **C**onnector. JDBC technology is an API (included in both J2SE and J2EE releases) that provides cross-DBMS connectivity to a wide range of SQL databases and access to other tabular data sources, such as spreadsheets or flat files. With a JDBC technology-enabled driver, you can connect all corporate data even in a heterogeneous environment.[20]
JNDI	**J**ava **N**aming and **D**irectory **I**nterface. The Java Naming and Directory Interface (JNDI) is part of the Java platform, providing applications based on Java technology with a unified interface to multiple naming and directory services. You can build powerful and portable directory-enabled applications using this industry standard.[21]
JRE	**J**ava **R**untime **E**nvironment.
JSP	**J**ava **S**erver **P**ages.
HTTP SESSION OBJECT	Hyper Text Transport Protocol (HTTP) Session Objects are used like cookies to maintain states as Web pages are considered stateless rather than stateful.

[19] Definition of JavaBean from the following Glossary Website: http://www.orafaq.com/glossary/faqglosj.htm, 7/17/04, Revision 2.1; Author: Frank Naudé.
[20] Definition from the following Sun Microsystems Website: http://java.sun.com/products/jdbc/.
[21] Definition from the following Sun Microsystems Website: http://java.sun.com/products/jndi/.

INSTITUTION	A Department of Veterans Affairs (VA) facility assigned a number by headquarters, as defined by Directive 97-058. An entry in the INSTITUTION file (#4) that represents the Veterans Health Administration (VHA). There are a wide variety of facility types in the INSTITUTION file, including medical centers, clinics, domiciliaries, administrative centers, Veterans Integrated Service Networks (VISNs), and so forth.
KAAJEE	Kernel Authentication and Authorization Java (2) Enterprise Edition.
KaajeeVistaLinkConnectionSpec	KAAJEE currently maintains this VistALink class and uses it to connect to the VistA M Server. This class extends VistaLinkConnectionSpecImpl. In other words, it inherits from the VistALink class VistaLinkConnectionSpecImpl. KAAJEE has added additional code in order to handle the IP address. **NOTE:** In the future, VistALink may incorporate and maintain this code.
MULTIDIVISIONAL	A facility is multidivisional if it supports one or more divisions. Health***e***Vet-VistA applications are required to be multidivisional-aware. Thus, it *must* be designed to work correctly at a multi-divisional facility.
ORACLE 9i	Oracle 9i (or higher version) is a relational database that supports the Structured Query Language (SQL), now an industry standard. Currently, it is used to store the KAAJEE SSPIs.
PATS	**P**atient **A**dvocate **T**racking **S**ystem. When completed, the Patient Advocate Tracking System will replace the current, site-based Patient Representative software with a national level application.
PRIMARY FACILITY	Primary facilities, also called Parent Facilities, are always medical centers, and they have a three-digit Station Number. A primary facility may be a standalone medical center, or it may be the parent facility of an integrated set of facilities, often called a healthcare network. For example, Palo Alto, CA is the headquarters of the Palo Alto Healthcare Network (HCN). Its Station Number is 640. An integrated set of facilities always falls within the boundary of a VISN.
PRODUCTION	A system on which *some* production (i.e., "live" data) is stored, accessed, and/or updated.
SINGLETON	"An object that cannot be instantiated. A singleton can be created, but it can't be instantiated by developers—meaning that the singleton class has control over how it is created. The restriction on the singleton is that there can be only one instance of a singleton created by the Java Virtual Machine (JVM)."[22]

[22] Definition taken from the "Java Coffee Break" Website: http://www.javacoffeebreak.com/articles/designpatterns/

SSL	**S**ecure **S**ocket **L**ayer. A low-level protocol that enables secure communications between a server and a browser. It provides communication privacy.
SSPI	**S**ecurity **S**ervice **P**rovider **I**nterfaces.
STATION NUMBER	A Station Number uniquely identifies every VA primary facility and division; however, entries for some facility types do not have Station Numbers. Station Numbers are stored in Field #99 in the VistA M Server INSTITUTION file (#4).
TEST	A system on which *no* production (i.e., "live" data) is stored, accessed, and/or updated.
TREEMAPS	TreeMaps are like name/value pairs. They are sorted by the keys. There are other types of maps as well (e.g., map, hashmap, hashtable, collection, etc.). TreeMaps have a Put and a Get method; therefore, you can use the Put method and pass in a key and an object. An object can be like any object (e.g., value object).
USER PROVISIONING	User account management—Create, modify, and delete user accounts and privileges (e.g., definition by roles and rules) for access to computer system resources. Enterprises typically use user provisioning to manage internal user access.[23]
VALUE OBJECT	Value Objects (VO) allow programs to store values for different elements where they can be extracted later using a method. They follow certain design patterns.
VISTALINK	VistALink is a transport layer between Java and M. VistALink consists of Java-side JCA adapter libraries and a MUMPS or M-side listener.
VPFS	**V**eterans **P**ersonal **F**inance **S**ystem. The re-hosted Integrated Patient Funds (IPF) software (a.k.a. Personal Funds of Patients [PFOP]) that is written in J2EE and planned to run on a centralized system. A Web browser front-end will be used for the user interface.
WAR (file)	**W**eb **AR**chive file (.war extension). Web Modules are packaged in .war files. A war file does not need to contain jsps and/or html content. A war file can be deployed by itself.

[23] Definition taken from the following Website: http://www.biu.ac.il/Computing/security/glossary%20of%20useful%20terms.htm (based on www. Gartner.com)

REF: For a comprehensive list of commonly used infrastructure- and security-related terms and definitions, please visit the Glossary Website:

http://vaww.vista.med.va.gov/iss/glossary.asp

For a comprehensive list of acronyms, please visit the Acronyms Website:

http://vaww.vista.med.va.gov/iss/acronyms/index.asp

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y