SINGLE SIGN-ON/USER CONTEXT (SSO/UC)

DEPLOYMENT GUIDE

Kernel Patch XU*8.0*337

September 2006

Department of Veterans Affairs
VistA Health Systems Design & Development (HSD&D)
Infrastructure & Security Services (ISS)

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.

Date	Revision	Description	Author(s)
09/27/06	1.0	Initial SSO/UC software and documentation release. **Kernel Patch XU*8.0*337**	ISS SSO/UC Development Team Oakland, CA and Bay Pines, FL OIFO: • Project Manager—Dan Soraoka • Project Planner—Laura Rowland • Developers—Alan Chan, Kyle Clarke, Wally Fort, Jose Garcia, Joel Ivey, and John Vrooland • SQA—Matt Alderman • Technical Writer—Thom Blom

Table i. Documentation revision history

Patch Revisions

For the current patch history related to this software, please refer to the Patch Module on FORUM.

NOTE: Kernel (i.e., Kernel Patch XU*8.0*337) is the designated custodial software package for SSO/UC-related software. However, SSO/UC comprises multiple patches and software releases from several VistA/Health*e*Vet-VistA applications.

REF: For the specific VistA M Server software patches required for the implementation of SSO/UC, please refer to Table 1-2 in Chapter 1, "SSO/UC Project Overview" in this manual.

Contents

Figures and Tables

Acknowledgements

The Single Sign-On/User Context (SSO/UC) Project Team consists of the following Development and Infrastructure Services (DaIS) and Infrastructure & Security Services (ISS) personnel (listed alphabetically within each category/title):

- ISS Program Manager—Larry Weldon
- ISS Project Manager—Dan Soraoka
- Centralized Planner Support Team (CPST)—Laura Rowland
- Developers—Alan Chan (KAAJEE), Kyle Clarke (VistALink), Wally Fort (Kernel), Jose Garcia (KAAJEE), Joel Ivey (RPC Broker), and John Vrooland (FatKAAT)
- Functional Analysts—Lauren Gorgoglione
- Software Quality Assurance (SQA)—Matt Alderman
- Technical Writer—Thom Blom

The SSO/UC Project Team would like to thank the following sites/organizations/personnel for their consultation and assistance in reviewing and/or testing SSO/UC-related software and documentation (listed alphabetically):

- Care Management (CM)/Health*e*Vet Development Team
- CCOW Team—Charles Arceneaux, Patrick (Tim) Landy, Dwyla Mosher, and David Tuma
- Computerized Patient Record System (CPRS) GUI Development Team
- Sentillion, Inc.—David Fusari, Karl Schoppe, and Eric Weaver
- Vitals Development Team

Orientation

This Deployment Guide is intended for use in conjunction with the Single Sign-On/User Context (SSO/UC) Project (Iteration 1). It outlines the details of SSO/UC-related software and gives guidelines on how the software is used within Veterans Health Information Systems and Technology Architecture (VistA).

The intended audience of this manual is all key stakeholders. The primary stakeholder is the VistA Infrastructure and Security Services (ISS). Additional stakeholders include:

- VistA application developers of CCOW-enabled applications under Health Systems Design & Development (HSD&D).
- Information Resource Management (IRM) and Information Security Officers (ISOs) at Veterans Affairs Medical Centers (VAMCs) responsible for computer management and system security.
- Enterprise VistA Support (EVS).
- VAMC personnel who will be using CCOW-enabled and SSO/UC-aware VistA Graphical User Interface (GUI) applications.

How to Use this Manual

This manual is divided into three major parts:

- User Guide—Provides general overview of the SSO/UC Project (Iteration 1).
- Developer Guide—Provides step-by-step instructions for VistA developers to follow and Application Program Interfaces (APIs) to use when writing CCOW-enabled and SSO/UC-aware applications.
- 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 SSO/UC software and the functionality it provides for Veterans Health Information Systems and Technology Architecture (VistA) software products.

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

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

Symbol	Description
ℹ	**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.

Table ii. Documentation symbol/term descriptions

- 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 boldface.
  - References to "**<Enter>**" within these snapshots indicate that the user should press the **Enter** key on the keyboard. Other special keys are represented within < > angle brackets. For example, pressing the **PF1** key can be represented as pressing **<PF1>**.
  - 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 and Delphi/Pascal 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).

How to Obtain Technical Information Online

Exported VistA M Server-based file, routine, and global documentation can be generated through the use of Kernel, MailMan, and VA FileMan utilities.

NOTE: Methods of obtaining specific technical information online will be indicated where applicable under the appropriate topic.

Help at Prompts

VistA M Server-based software provides online help and commonly used system default prompts. Users are encouraged to enter question marks at any response prompt. At the end of the help display, you are immediately returned to the point from which you started. This is an easy way to learn about any aspect of VistA M Server-based software.

Obtaining Data Dictionary Listings

Technical information about VistA M Server-based files and the fields in files is stored in data dictionaries (DD). You can use the List File Attributes option on the Data Dictionary Utilities submenu in VA FileMan to print formatted data dictionaries.

REF: For details about obtaining data dictionaries and about the formats available, please refer to the "List File Attributes" chapter in the "File Management" section of the *VA FileMan Advanced User Manual*.

Assumptions About the Reader

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

- VistA/HealtheVet-VistA computing environment:
  - Kernel—VistA M Server software
  - Remote Procedure Call (RPC) Broker—VistA Client/Server software
  - VA FileMan data structures and terminology—VistA M Server software
  - VistALink—VistA M Server and client workstation software
- Microsoft Windows environment
- M programming language
- Object Pascal programming language/Borland Delphi Integrated Development Environment (IDE)—RPC Broker
- Java programming language/Java Development Kit (JDK)—VistALink
- CCOW and CCOW Standard—Sentillion Vergence Context Vault and Desktop Components

This manual provides an overall explanation of configuring SSO/UC and the functionality contained in SSO/UC-related software (designated owner Kernel Patch XU*8.0*337). However, no attempt is made to explain how the overall VistA and HealtheVet-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 VistA and HealtheVet-VistA. For example, go to the Veterans Health Administration (VHA) Office of Information (OI) Health Systems Design & Development (HSD&D) Home Page at the following Intranet Web address:

http://vista.med.va.gov/

Reference Materials

Readers who wish to learn more about the SSO/UC-related software should consult the following:

- *Single Sign-On/User Context (SSO/UC) Installation Guide (Kernel Patch XU*8.0*337)*
- *Single Sign-On/User Context (SSO/UC) Deployment Guide (Kernel Patch XU*8.0*337)*, this manual
- SSO/UC Web site: http://vaww.vista.med.va.gov/kernel/sso/index.asp
- *Kernel Systems Manual (Version 8.0)*
- *RPC Broker Installation Guide (XWB*1.1*40)*
- *RPC Broker Developer's Guide (online help, XWB*1.1*40)*
- *RPC Broker Getting Started with the RPC Broker Development Kit (BDK, XWB*1.1*40)*

- *RPC Broker Systems Manual (XWB*1.1*40)*
- *VistALink Installation Guide (Version 1.5)*
- *VistALink Developer/System Manager Manual (Version 1.5)*
- *VistALink Technical Manual and Package Security Guide (Version 1.5)*

> **REF:** For more information on VistALink, please refer to the Application Modernization Foundations Web site located at the following Web address:
>
> http://vaww.vista.med.va.gov/migration/foundations/vl/index.htm

- *Sentillion Vergence User Link Installation Instructions*
- *Sentillion Vergence Context Vault User's Guide (Version 3.3)*
- *Sentillion Vergence Desktop Components Installation Guide (Version 3.3)*

VistA/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 (i.e., ACROREAD.EXE), which is freely distributed by Adobe Systems Incorporated at the following Web address:

http://www.adobe.com/

> **REF:** For more information on the use of the Adobe Acrobat Reader, please refer to the Adobe Acrobat Quick Guide at the following Web address:
>
> http://vista.med.va.gov/iss/acrobat/index.asp

VistA/Health*e*Vet-VistA documentation can be downloaded from the Health Systems Design and Development (HSD&D) VistA Documentation Library (VDL) Web site:

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

VistA/Health*e*Vet-VistA documentation and software can also be downloaded from the Enterprise VistA Support (EVS) anonymous directories:

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
Preferred Method	download.vista.med.va.gov

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

DISCLAIMER: The appearance of any external hyperlink references in this manual does not constitute endorsement by the Department of Veterans Affairs (VA) of this Web site 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 the SSO/UC-related software. It is intended for use in conjunction with the SSO/UC Project (Iteration 1). It details the user-related SSO/UC documentation (e.g., overview of the SSO/UC Project [Iteration 1], management of SSO/UC-related software, etc.).

1. SSO/UC Project Overview

Introduction

The Veterans Health Administration (VHA) information systems user community has expressed a need for a single sign-on (SSO) service with interfaces to VistA, Health*e*Vet VistA, and non-VistA systems. The goal of the Single Sign-on/User Context (SSO/UC) Project (Iteration 1) is to address this need by providing a secure single sign-on architecture. This architecture allows users to authenticate and sign on to multiple applications that are CCOW-enabled and SSO/UC-aware using a single set of credentials, which will reduce the need for multiple IDs and passwords in the VistA/Health*e*Vet-VistA clinician desktop environment.

SSO capability is implemented within the framework of the HL7 CCOW User Context standard. The HL7 CCOW User Context standard is:

- A standard of the HL7 standards body.
- Provides coordination among client healthcare applications, allowing them to synchronize around several CCOW subjects and share context (e.g., Patient, Encounter, and User Context).
- Ensures secure and consistent context management (e.g., Patient, Encounter, and User Context):

> "By synchronizing and coordinating applications so that they automatically follow the user's context, the CCOW Standard serves as the basis for ensuring secure and consistent access to patient information from heterogeneous sources. The benefits include applications that are easier to use, increased utilization of electronically available information, and an increase in patient safety. Further, CCOW support for secure context management provides a healthcare standards basis for addressing HIPAA requirements. For example, CCOW enables the deployment of highly secure single sign-on solutions."[1]

While User Context is not a full SSO solution, it provides some attractive features of SSO. Though a full SSO solution may be something desired by the VHA, it's beyond the scope of this project iteration. This project is the first step toward providing a full SSO solution.

The SSO/UC Project (Iteration 1) software follows a prototype demonstration project (i.e., SSO/UC Prototype). The prototype uncovered some basic and essential features of a security architecture for authenticating and authorizing users in VA system environments, which have been incorporated into the SSO/UC Project (Iteration 1) software.

REF: For more information on the SSO/UC Prototype, please refer to "Appendix A—SSO/UC Prototype (Pre-release)," in this manual.

[1] From a white paper titled "Overview of HL7's CCOW Standard"; copyrighted 2001 Health Level Seven, Inc.; author Robert Seliger (Sentillion), Co-Chair, CCOW Technical Committee from the http://www.hl7.org/special/committees/visual/visual.cfm Web site.

This manual discusses in more detail the various software applications/modules that, together, provide for SSO/UC functionality:

Application/Module	Location	Description
VistA M Server	VistA M Server	(required) This is the "backend server" where the Kernel software acts as the authentication source for all VistA applications (i.e., client/server and roll-and-scroll applications). **REF:** For a list of Vista M Server patches, please refer to Table 1-2.
Sentillion Vergence Context Vault	Server	(required) Sentillion's Vergence Context Vaults Components Commercial-Off-The-Shelf (COTS) software (see Table 6-5)—Currently in place in the VA to support CCOW subjects including Patient Context, is the basis for the implementation of CCOW User Context-based SSO on the server. The Sentillion Vergence Context Vault maintains the CCOW User Context state for each user at a facility. **NOTE:** This COTS software is already purchased, being installed, and configured as part of the Clinical Context Management Project (CCOW) release for Patient Context. However, for SSO/UC, a separate User subject license is required. This license has already been purchased, but it now *must* be installed. **REF:** For more information on the CCOW Package Release, please refer to the Context Management Project (CCOW) release documentation available on the EVS Anonymous Directories.
Sentillion Vergence Desktop Components	Client	(required for developer workstations only) Sentillion's Vergence Desktop COTS software (see Table 6-5)—Currently in place in the VA to support CCOW subjects including Patient Context, is the basis for the implementation of CCOW User Context-based SSO on the client workstation. The Sentillion Vergence Desktop Components provide communication/linkage between the CCOW-enabled login components embedded in the CCOW-enabled VistA applications on the client workstation and the Sentillion Vergence Context Vault on the server. **NOTE:** This COTS software is already purchased, being installed, and configured as part of the Clinical Context Management Project (CCOW) release for Patient Context. **REF:** For more information on the CCOW Package Release, please refer to the Context Management Project (CCOW) release documentation available on

Application/Module	Location	Description
		the EVS Anonymous Directories.
Rich Client Login Components/Classes: • RPC Broker • VistALink	Client	(required for developer workstations only) The RPC Broker (COM-based) login component and VistALink (Java-based) login classes allow rich client/server applications to authenticate against the VistA M Server and obtain a persistent connection over which remote procedure calls (RPCs) are executed. Both of these components/classes are modified in the SSO/UC Project (Iteration 1) to be CCOW-enabled and SSO/UC-aware. CCOW-enabled applications (e.g., Care Management [CM], Computerized Patient Record System [CPRS], Vitals) *must* invoke these modified login components/classes to also become SSO/UC-aware, and post-login, are expected to follow a set of rules when listening for subsequent context changes for the User subject.
CCOW Context Monitor	Client	(optional for all client workstations) This is a (test) rich client application that runs in the background and is automatically started at system startup. Users will know the CCOW Context Monitor is running by the icon displayed in the Windows taskbar status area (a.k.a. system tray, see Figure 2-3). This application provides the current CCOW User Context identity and the ability to clear the User Context.

Table 1-1. SSO/UC-related software applications/modules

REF: For the specific software patches required for the implementation of SSO/UC, please refer to Table 1-2 in this chapter.

REF: For more detailed information on these SSO applications/modules, please refer to Chapter 2, "SSO/UC VistA Applications/Modules," in this manual.

Features

Single Sign-On/User Context (SSO/UC) Project (Iteration 1) provides the following features and functionality:

- Secure single set of passwords with which to authenticate and sign on to multiple CCOW-enabled and SSO/UC-aware applications (e.g., Care Management [CM], Computerized Patient Record System [CPRS], Vitals) using the same VistA M Server for authentication and backend services (i.e., Kernel).
- SSO capability implemented within the framework of HL7 CCOW User Context Standard.
- Compatible technology that can be easily migrated toward any future Authentication and Authorization (AA) development efforts.

- Supported computing environments/application types:
  - VistA Rich Client/M Server Applications:
    - RPC Broker-based, CCOW-enabled applications (i.e., COM client applications developed in Borland Delphi).
    - VistALink-based, CCOW-enabled applications (i.e., Java client applications).
  - Citrix Server Environment—VistA rich client, CCOW-enabled applications stored on a Citrix Server.

Architectural Scope

The architectural scope of the SSO/UC Project (Iteration 1) is as follows:

- **Use of Kernel Authentication—**Kernel is used as the authenticator. The Kernel CCOW login token is a valid means of authenticating on a backend VistA M Server.
- **Use of Sentillion Vergence Desktop Components and Context Vault—**SSO/UC is implemented by using Sentillion's Vergence Desktop Components on the client workstation and the Sentillion Vergence Context Vault on the server to communicate and store user sign-on credentials in a CCOW User Context, in a similar fashion to how VA is already using the Vergence Desktop Components and Context Vault to synchronize applications on Patient Context. It *must* be properly configured for User Context, including the required application names and passcodes.
- **Rich Client/M Server-based Application Support—**This document only discusses the CCOW SSO/UC functionality provided with VistA rich client/server clinical healthcare applications. However, the Health*e*Vet-VistA Kernel Authentication and Authorization for Java 2 Enterprise Edition (KAAJEE) software for Java Web-based applications and the Fat-client Kernel Authentication and Authorization Tool (FatKAAT) software for rich clients, both with an application server middle tier (e.g., BEA WebLogic), will also be made CCOW-enabled and SSO/UC-aware.

  > **REF:** For more information on KAAJEE, please refer to the *Kernel Authentication & Authorization for Java 2 Enterprise Edition (KAAJEE) Installation Guide* and *KAAJEE Deployment Guide*.
  >
  > For more information on FatKAAT, please refer to the *Fat-client Kernel Authentication & Authorization Tool (FatKAAT) Installation Guide* and *FatKAAT Deployment Guide*.

- **Single Facility SSO/UC Support—**The current Kernel CCOW login token scheme is only valid at a single VistA M Server; the "chain of trust" provided by the Kernel does *not* span VistA M Servers. Thus, the CCOW-enabled SSO/UC is *not* supported across facility boundaries in the SSO/UC Project (Iteration 1).
- **User Context Switching *Not* Supported—**The CCOW User Context standard is designed to support changing users (i.e., User Context switching) without closing applications. However, VA policy requires the application user to be the client workstation user. Therefore, User Context switching will *not* be supported in the SSO/UC Project (Iteration 1).
- **Kiosk-style Workstations *Not* Supported—**Kiosk-mode is a particularly effective operational model for workstations in clinical ward areas, where many users need to rapidly perform clinical-

related computing on a single workstation (i.e., where multiple users can log into and out of the same application without terminating the application between users). CCOW User Context switching can be enabled in a kiosk-mode. However, VA policy requires the application user to be the client workstation user. Therefore, kiosk-mode will *not* be supported in the SSO/UC Project (Iteration 1).

REF: For more information on Kiosk-style workstations, please refer to the *Single Sign-on Iteration 1 Software Design Document (SDD)*.

Dependencies—VistA M Server Patches

Kernel (i.e., Kernel Patch XU*8.0*337) is the designated custodial software package of the Infrastructure & Security Services (ISS) SSO/UC-related software. However, SSO/UC comprises multiple patches from several VistA applications (listed by category and software name):

Category	Software	Version	Patch	Subject/Description
Client (for development)	RPC Broker	1.1	XWB*1.1*40	BDK32 With TCCOWRPCBroker (released on FORUM)—This client-side patch updates the Broker Development Kit (BDK). It allows developers to make their CCOW-enabled RPC Broker-based rich client applications SSO/UC-aware via the TCCOWRPCBroker component. It also enables the TRPCBroker and TCCOWRPCBroker components to establish a connection with a non-callback server (as provided with Broker Patch XWB*1.1*35). **NOTE:** This client-side patch is dependent on the server-side RPC Broker Patch XWB*1.1*35.
Server	Kernel	8.0	XU*8.0*265	3 Strikes and You Are Out (released on FORUM)—This patch enhances security by providing IP address locking functionality (terminal servers are uniquely handled). Also provides special locking security for individual users. **NOTE:** This patch is required for Kernel Patch XU*8.0*337.
			XU*8.0*284	API for Production Account Check (released on FORUM)—This patch adds two parameters to XUP with SYS or USR values that can be set to control XUP. It added the $$PROD^XUPROD() API.

Category	Software	Version	Patch	Subject/Description
			XU*8.0*337	CCOW SSO/UC Support (released on FORUM)—This patch updates Kernel authentication and authorization routines in order to enable SSO/UC and provide the VPID for SSO/UC. It also distributes the XUS ALLKEYS RPC and adds the GUI POST SIGN-ON field (#231) to the KERNEL SYSTEM PARAMETERS file (#8989.3). **NOTE:** Kernel (i.e., Kernel Patch XU*8.0*337) is the designated custodial software package of the SSO/UC-related software. This patch is dependent on Kernel Patch XU*8.0*265, because Kernel Patches XU*8.0*265 and 337 are modifying the same Kernel authentication and authorization routines. Also, Kernel Patch XU*8.0*284 (released), though not officially part of the SSO/UC Project (Iteration 1), contains an API whose need arose in discussions with developers during the SSO/UC Project (Iteration 1).
			XU*8.0*361	Proxy Application User for Re-hosting Effort (released on FORUM)—SSO/UC uses the Application Proxy user provided with this patch.

Category	Software	Version	Patch	Subject/Description
	RPC Broker	1.1	XWB*1.1*35	NON-callback Server (released on FORUM)—This patch provides local sites with the ability to control the range of ports used in connecting to joint and/or contracting facilities, useful behind firewalls. This patch contains the following: • Modified the XWB LISTENER STARTER option. • Added the XWB LISTENER STOP ALL option. • Modified the RPC BROKER SITE PARAMETERS file (#8994.1). • Modified the XWB LISTENER EDIT template. • Entry added to the PARAMETER DEFINITION file (#8989.51). • Modified/New routines. **NOTE:** This server-side patch is required for client-side RPC Broker Patch XWB*1.1*40.

Table 1-2. Dependencies—VistA M Server patches

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

NOTE: This table only includes VistA M Server software patches required for SSO/UC; it does *not* list COTS software or other VistA/Health*e*Vet-VistA software/patches that are not directly related to SSO/UC.

REF: For a list of the Commercial-Off-The-Shelf (COTS) software required for the SSO/UC Project (Iteration 1), please refer to Table 6-5 in the "External Relations" chapter in this manual.

Process Overview

After logging onto the NT Network, when a user launches the first CCOW-enabled and SSO/UC-aware application of their session, he/she will be presented with the normal interactive Access/Verify code dialogue box. Only after the user has successfully signed onto their first CCOW-enabled and SSO/UC-aware application is the User Context available for use with other CCOW-enabled and SSO/UC-aware applications, and thus, allowing single sign-on (SSO).

As with a standard login, the CCOW-enabled and SSO/UC-aware login logic is performed by the RPC Broker login component or VistALink login classes embedded in the client/server application, and *not* by the application itself.

For the set of client/server applications that are CCOW-enabled, the task is to achieve SSO for a new client application that is attempting to join the CCOW User Context and establish its own, new, connection to a VistA M Server. To achieve SSO through CCOW, when the resource to be protected is a new session to the VistA M Server, something in the CCOW context needs to be passed that allows another application to authenticate a new session/connection with Kernel on the VistA M Server, without prompting the user for their credentials again (i.e., Access and Verify codes).

VistA User Context sharing (or "User linking") relies on an initial authentication via Kernel's NEW PERSON file (#200). Thus, when a user starts up the first CCOW-enabled and SSO/UC-aware RPC Broker- or VistALink-based rich client application on a client workstation to access a VistA M Server, the user *must* initially authenticate against Kernel on that VistA M Server by entering their Access and Verify codes. This authentication produces a Kernel CCOW login token that is stored on the VistA M Server. The User Context is stored in the Sentillion Vergence Context Vault for subsequent use for connection to other CCOW-enabled and SSO/UC-aware applications.

Once a CCOW-enabled and SSO/UC-aware application is launched and has successfully shared User Context with the Context Vault, subsequent CCOW-enabled and SSO/UC-aware applications running on the same client workstation can then connect to the same VistA M Server using the stored User Context and Kernel CCOW login token without requiring the user to re-enter their Access and Verify codes (i.e., performs a *non*-interactive logon). This allows the user to only have to sign on once to a particular VistA M Server.

For security reasons, the Kernel CCOW login token is only valid for a user for a prescribed period of time. Once the User Context has been cleared from the Context Vault and all CCOW-enabled and SSO/UC-aware applications started by that user on a client workstation are closed, the Kernel CCOW login token is removed from the VistA M Server via Kernel utilities.

REF: For more information on the Kernel CCOW login token expiration period, please refer to the "Kernel CCOW Login Token Expiration" topic in Chapter 5, "Implementation and Maintenance," in this manual.

The overall SSO/UC functionality can be described as three separate processes:

1. **Initial CCOW-enabled and SSO/UC-aware Application Startup—**User starts the first CCOW-enabled and SSO/UC-aware application on a client workstation. Initially, the user *must* authenticate to Kernel and a Kernel CCOW login token is stored as part of the CCOW User Context.
2. **Subsequent CCOW-enabled and SSO/UC-aware Application Startup (*Valid* Token)—**User already authenticated to Kernel, has a *valid* Kernel CCOW login token, and starts subsequent CCOW-enabled and SSO/UC-aware application on a client workstation.
3. **Subsequent CCOW-enabled and SSO/UC-aware Application Startup (*Invalid/Expired* Token)—**User already authenticated to Kernel, has an *invalid/expired* Kernel CCOW login token, and starts subsequent CCOW-enabled and SSO/UC-aware application on a client workstation.

NOTE: These processes assume that the user has already logged into NT on the client workstation.

Initial CCOW-enabled and SSO/UC-aware Application Startup

The CCOW-enabled and SSO/UC-aware RPC Broker login component or VistALink login classes, embedded in the initial CCOW-enabled and SSO/UC-aware RPC Broker- or VistALink-based rich client/server application, connect to the IP address and listener port of the target VistA M Server.

The login component/classes then look for a CCOW Context Manager on the client workstation (i.e., Sentillion Vergence Desktop Components) and verify if a CCOW User Context has already been established and is stored in the Sentillion Vergence Context Vault.

Since this is the initial CCOW-enabled application, no User Context has been established yet. Thus, the login component/classes *must* initially authenticate the user against Kernel on the VistA M Server by prompting the user to enter their Access and Verify codes (i.e., performs a normal login).

ℹ **NOTE:** *Prior* to authentication, the CCOW GUI icon will show that the application is *not* in User Context.

The login component/classes then make a call to the VistA M Server with the user credentials (i.e., Access and Verify codes), which successfully logs the user onto VistA.

Kernel on the VistA M Server generates a Kernel CCOW login token that is cached, along with the creation time, on the VistA M Server (temporary global) and returned to the login component/classes on the client workstation.

The login component/classes then use that token along with the system identifier and user name to create a new CCOW User Context, which is stored in the Sentillion Vergence Context Vault. In order to set context, the login component/classes *must* authenticate to the Context Vault via an application name and passcode.

ℹ **NOTE:** *After* authentication, the CCOW GUI icon should show that the application is now in User Context.

Once the User Context has been established, subsequent CCOW-enabled and SSO/UC-aware applications can join the CCOW User Context without prompting the user for an Access and Verify code.

This process is shown in greater detail in the figure that follows:

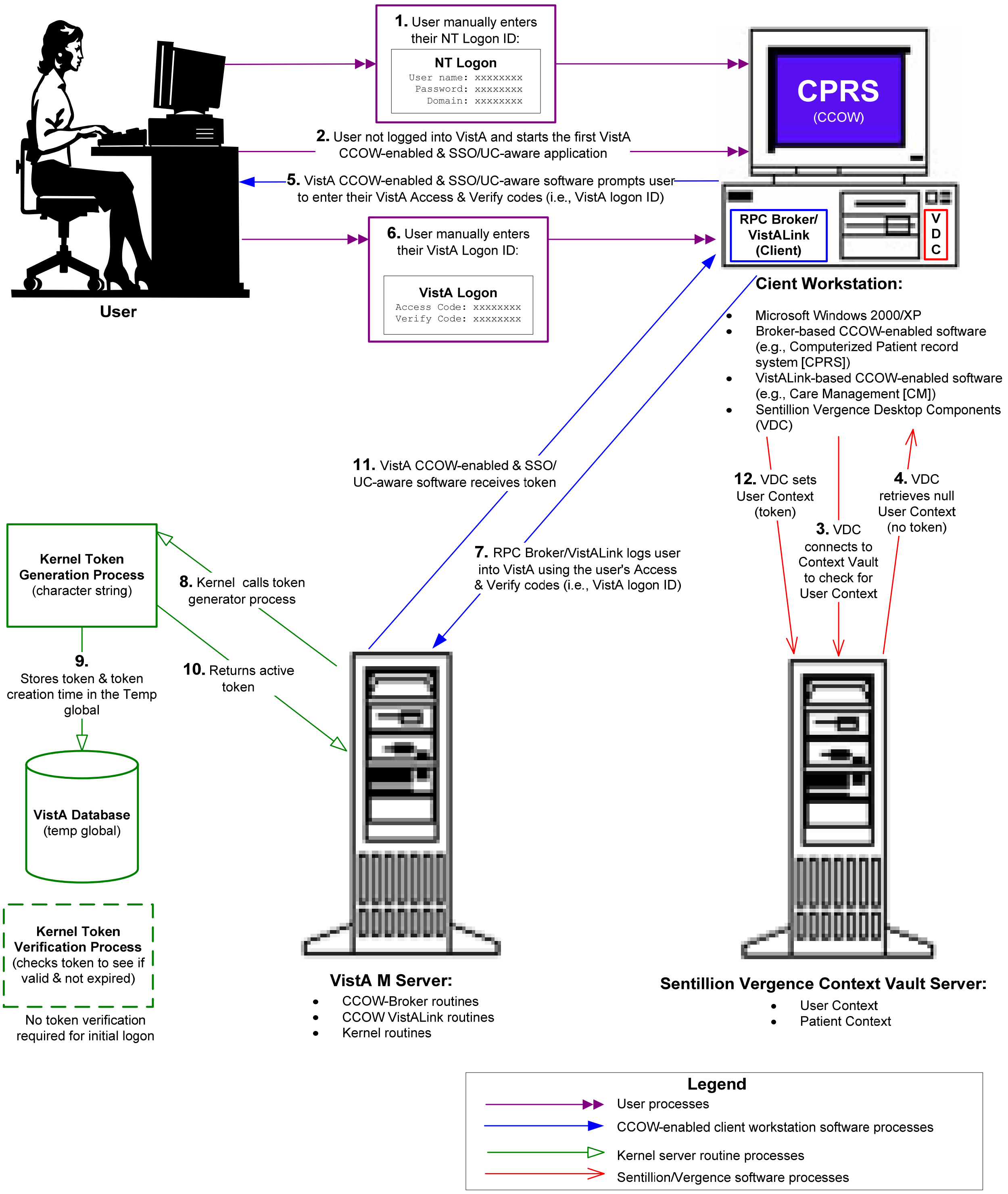

1. User manually enters their NT Logon ID:
NT Logon
User name: xxxxxxxx
Password: xxxxxxxx
Domain: xxxxxxxx
CPRS
(CCOW)
2. User not logged into VistA and starts the first VistA CCOW-enabled & SSO/UC-aware application
5. VistA CCOW-enabled & SSO/UC-aware software prompts user to enter their VistA Access & Verify codes (i.e., VistA logon ID)
6. User manually enters their VistA Logon ID:
VistA Logon
Access Code: xxxxxxxx
Verify Code: xxxxxxxx
RPC Broker/ VistALink (Client)
V D C
User
Cient Workstation:
Microsoft Windows 2000/XP
Broker-based CCOW-enabled software (e.g., Computerized Patient record system [CPRS])
VistALink-based CCOW-enabled software (e.g., Care Management [CM])
Sentillion Vergence Desktop Components (VDC)
11. VistA CCOW-enabled & SSO/ UC-aware software receives token
12. VDC sets User Context (token)
4. VDC retrieves null User Context (no token)
3. VDC connects to Context Vault to check for User Context
7. RPC Broker/VistALink logs user into VistA using the user's Access & Verify codes (i.e., VistA logon ID)
Kernel Token Generation Process (character string)
8. Kernel calls token generator process
9. Stores token & token creation time in the Temp global
10. Returns active token
VistA Database (temp global)
Kernel Token Verification Process (checks token to see if valid & not expired)
No token verification required for initial logon
VistA M Server:
CCOW-Broker routines
CCOW VistALink routines
Kernel routines
Sentillion Vergence Context Vault Server:
User Context
Patient Context
Legend
User processes
CCOW-enabled client workstation software processes
Kernel server routine processes
Sentillion/Vergence software processes

Figure 1-1. SSO/UC & CCOW Process Overview:
User performs initial VistA **logon and starts first CCOW-enabled and SSO/UC-aware application.**

Subsequent CCOW-enabled and SSO/UC-aware Application Startup (*Valid* Token)

The CCOW-enabled and SSO/UC-aware RPC Broker login component or VistALink login classes, embedded in the subsequent CCOW-enabled and SSO/UC-aware RPC Broker- or VistALink-based rich client/server application, connect to the IP address and listener port of the target VistA M Server.

The login component/classes then look for a CCOW Context Manager on the client workstation (i.e., Sentillion Vergence Desktop Components) and verify if a CCOW User Context has already been established and is stored in the Sentillion Vergence Context Vault.

Since this is a subsequent CCOW-enabled and SSO/UC-aware application, a User Context and Kernel CCOW login token have already been established. Thus, the login component/classes make a call to the VistA M Server to validate the Kernel CCOW login token.

ℹ **NOTE:** *Prior* to authentication, the CCOW GUI icon will show that the application is *not* in User Context.

Kernel validates the token by checking the token creation time against the current system time and if the token has *not* expired, based on the parameter set in the CCOW TOKEN TIMEOUT field (#30.1) in the KERNEL SYSTEM PARAMETERS file (#8989.3), the token is valid. Since this is a *valid* Kernel CCOW login token described in this process, the logon is successful and the application joins the CCOW User Context without prompting the user for an Access and Verify code. Whatever division selection (if any) the user made on the initial login is restored for the subsequent login.

ℹ **NOTE:** *After* authentication, the CCOW GUI icon should show that the application is now in User Context.

This process is shown in greater detail in the figure that follows:

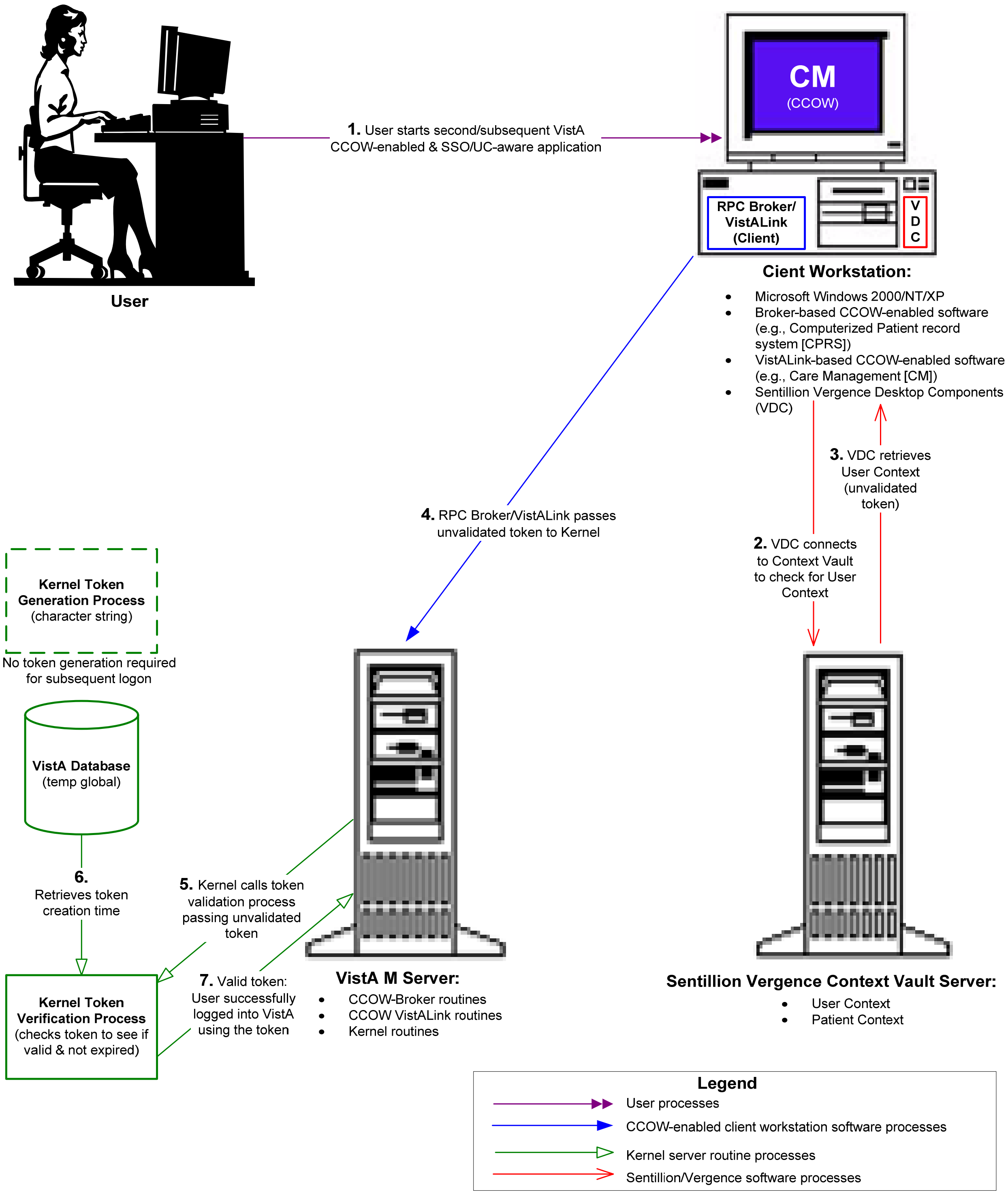

User
CM
(CCOW)
RPC Broker/ VistALink (Client)
V
D
C
1. User starts second/subsequent VistA CCOW-enabled & SSO/UC-aware application
Cient Workstation:
Microsoft Windows 2000/NT/XP
Broker-based CCOW-enabled software (e.g., Computerized Patient record system [CPRS])
VistALink-based CCOW-enabled software (e.g., Care Management [CM])
Sentillion Vergence Desktop Components (VDC)
2. VDC connects to Context Vault to check for User Context
3. VDC retrieves User Context (unvalidated token)
4. RPC Broker/VistALink passes unvalidated token to Kernel
Kernel Token Generation Process (character string)
No token generation required for subsequent logon
VistA Database (temp global)
6. Retrieves token creation time
5. Kernel calls token validation process passing unvalidated token
7. Valid token: User successfully logged into VistA using the token
Kernel Token Verification Process (checks token to see if valid & not expired)
VistA M Server:
CCOW-Broker routines
CCOW VistALink routines
Kernel routines
Sentillion Vergence Context Vault Server:
User Context
Patient Context
Legend
User processes
CCOW-enabled client software processes
Kernel server routine processes
Sentillion/Vergence software processes

**Figure 1-2. SSO/UC & CCOW Process Overview:
User signed onto VistA, valid token, and starts subsequent CCOW-enabled and SSO/UC-aware application.**

Subsequent CCOW-enabled and SSO/UC-aware Application Startup (*Invalid/Expired* Token)

The CCOW-enabled and SSO/UC-aware RPC Broker login component or VistALink login classes, embedded in the subsequent CCOW-enabled and SSO/UC-aware RPC Broker- or VistALink-based rich client/server application, connect to the IP address and listener port of the target VistA M Server.

The login component/classes then look for a CCOW Context Manager on the client workstation (i.e., Sentillion Vergence Desktop Components) and verify if a CCOW User Context has already been established and is stored in the Sentillion Vergence Context Vault.

Since this is a subsequent CCOW-enabled application, a User Context and Kernel CCOW login token have already been established. Thus, the login component/classes make a call to the VistA M Server to validate the Kernel CCOW login token.

Kernel validates the token by checking the token creation time against the current system time and if the token has *not* expired, based on the parameter set in the CCOW TOKEN TIMEOUT field (#30.1) in the KERNEL SYSTEM PARAMETERS file (#8989.3), the token is valid. Since this is an *invalid/expired* Kernel CCOW login token described in this process, the logon fails. Thus, the login component/classes *must* re-authenticate the user against Kernel on the VistA M Server by prompting the user to re-enter their Access and Verify codes (i.e., performs a normal login).

NOTE: *Prior* to authentication, the CCOW GUI icon will show that the application is *not* in User Context.

The login component/classes then make a call to the VistA M Server with the re-entered user credentials (i.e., Access and Verify codes), which successfully logs the user onto VistA.

NOTE: *After* authentication, the CCOW GUI icon should show that the application is now in User Context.

Since no new Kernel token is created and stored in this process, any subsequent CCOW-enabled and SSO/UC-aware or other applications started by this same user on this same client workstation will require the user to re-authenticate to the Kernel M Server via their Access and Verify codes each time. In other words, the SSO/UC functionality will no longer be available to a user once the Kernel token associated with that user is invalidated/expired.

This process is shown in greater detail in the figure that follows:

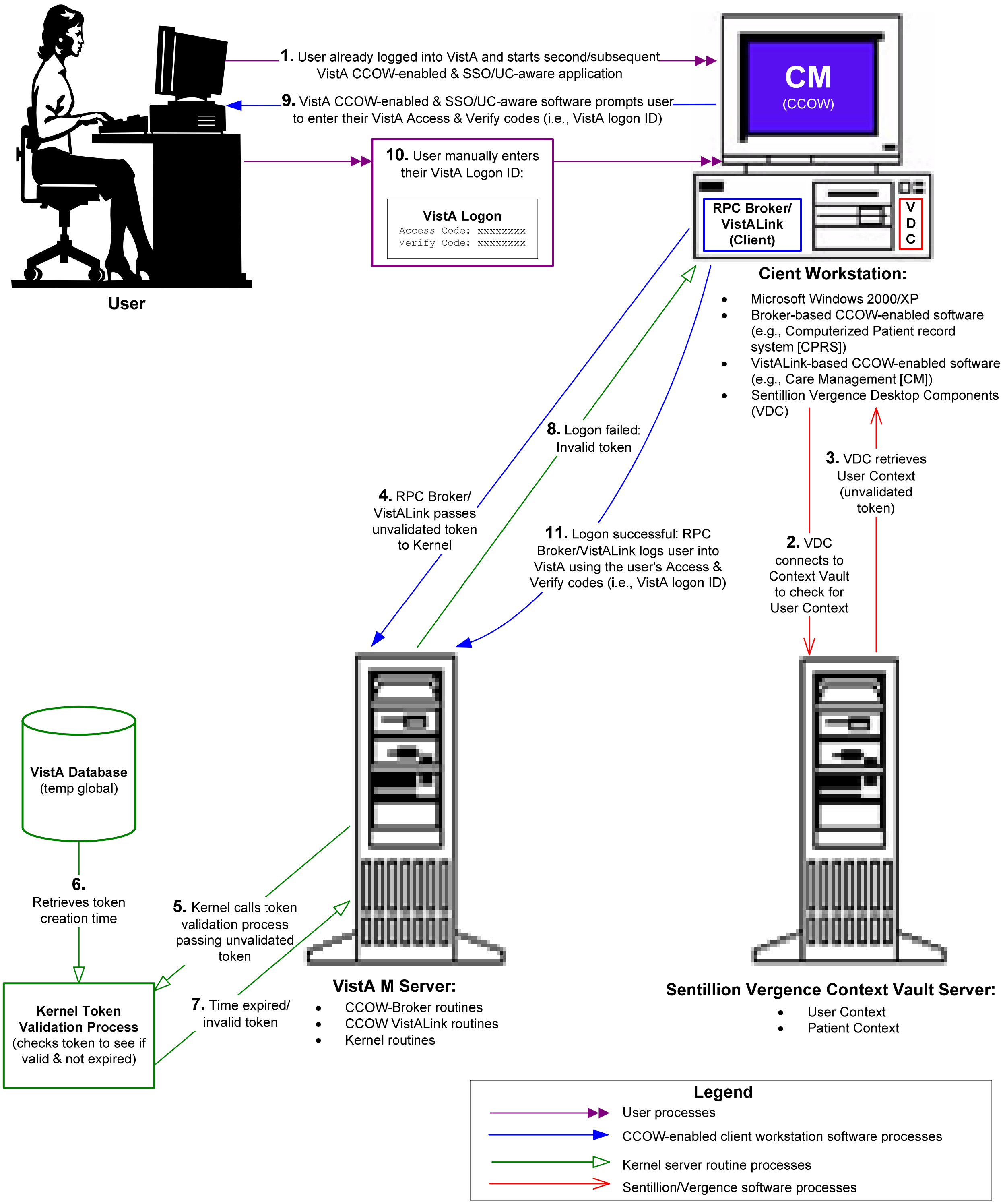

1. User already logged into VistA and starts second/subsequent VistA CCOW-enabled & SSO/UC-aware application
9. VistA CCOW-enabled & SSO/UC-aware software prompts user to enter their VistA Access & Verify codes (i.e., VistA logon ID)
10. User manually enters their VistA Logon ID:
VistA Logon
Access Code: xxxxxxxx
Verify Code: xxxxxxxx
User
CM
(CCOW)
RPC Broker/ VistALink (Client)
V D C
Cient Workstation:
Microsoft Windows 2000/XP
Broker-based CCOW-enabled software (e.g., Computerized Patient record system [CPRS])
VistALink-based CCOW-enabled software (e.g., Care Management [CM])
Sentillion Vergence Desktop Components (VDC)
8. Logon failed: Invalid token
4. RPC Broker/ VistALink passes unvalidated token to Kernel
11. Logon successful: RPC Broker/VistALink logs user into VistA using the user's Access & Verify codes (i.e., VistA logon ID)
3. VDC retrieves User Context (unvalidated token)
2. VDC connects to Context Vault to check for User Context
VistA Database (temp global)
6. Retrieves token creation time
5. Kernel calls token validation process passing unvalidated token
7. Time expired/ invalid token
Kernel Token Validation Process (checks token to see if valid & not expired)
VistA M Server:
CCOW-Broker routines
CCOW VistALink routines
Kernel routines
Sentillion Vergence Context Vault Server:
User Context
Patient Context
Legend
User processes
CCOW-enabled client workstation software processes
Kernel server routine processes
Sentillion/Vergence software processes

Figure 1-3. SSO/UC & CCOW Process Overview:
User signed onto VistA, invalid token, and starts subsequent CCOW-enabled and SSO/UC-aware application.

2. SSO/UC VistA Applications/Modules

The chapter describes the new or modified functionality made to the SSO/UC-related software applications/modules as listed in Table 1-1 in Chapter 1 in this manual.

A CCOW-enabled and SSO/UC-aware VistA application is an application that has been re-compiled using the CCOW-enabled RPC Broker login component or VistALink login classes. SSO/UC capability comes into play when you are using an SSO/UC-aware application (e.g., Care Management or CPRS). Additionally, any applications utilizing the CCOW-enabled SSO/UC functionality will need to be modified to follow the business rules associated with synchronizing the CCOW User subject.

REF: For information on how to modify a VistA CCOW-enabled rich client application to make it SSO/UC-aware, please refer to Chapter 4, "Making VistA Applications SSO/UC-aware," in this manual.

Kernel—Authentication Interface to VistA

Authentication is the process of verifying a user identity to ensure that the person requesting access to a clinical information system is, in fact, that person to whom entry is authorized. For the SSO/UC Project (Iteration 1), after a user has been initially authenticated to Kernel on the VistA M Server via their Access and Verify codes, a Kernel CCOW login token will be created and stored on the VistA M Server and User Context stored in the Sentillion Vergence Context Vault. The Kernel CCOW login token will then be used to authenticate and authorize any subsequent CCOW-enabled and SSO/UC-aware application started up by the same user, on the same client workstation, and accessing the same VistA M Server.

The following User Context data was identified to support single sign-on in the SSO/UC final solution:

```
//The VistA Domain
CCOW_LOGON_ID = 'user.id.logon.vistalogon';        //CCOW
//The VistA Token
CCOW_LOGON_TOKEN = 'user.id.logon.vistatoken';     //CCOW
//The VistA user Name
CCOW_LOGON_NAME = 'user.id.logon.vistaname';       //CCOW
//The User Name in VistA
CCOW_NAME_VistA = 'user.id.logon.vpid';
```

Figure 2-1. User Context data for SSO/UC Prototype solution

NOTE: The Kernel CCOW login token will also contain the Department of Veterans Affairs Personal Identification (VPID), which will be null until the VA enterprise use of VPID is enacted. The VPID will be retrieved from the NEW PERSON file (#200), where it will be stored in addition to being stored in national directories. There will also be a VPID entry stored in the Sentillion Vergence Context Vault.

Kernel software on the VistA M Server is the approved method of authentication for all users in the VHA environment. Kernel was assessed as the most straightforward and timely approach to also be used for single sign-on authentication in the SSO/UC Project (Iteration 1). By using Kernel as the authenticator for SSO, the NEW PERSON file (#200) continues to serve as the single user data store for VistA, the SSO/UC Project (Iteration 1), KAAJEE sub-project (e.g., J2EE Web applications), and FatKAAT Project (e.g., J2EE rich client applications).

A single user data store provides the following benefits:

- Ease of coding requirements by application developers.
- Ease of file maintenance by IRM.
- Avoids an additional user store, which simplifies the migration to any future AA solutions.

The Kernel SSO/UC functionality was introduced with Kernel Patch XU*8.0*337 (server-side).

RPC Broker

The RPC Broker connects Borland Delphi-based rich client COM applications running on a Microsoft Windows client workstation to a VistA M Server. This connection allows data retrieval from the VistA M database. The RPC Broker login component uses the Access and Verify codes to authenticate a user to VistA.

For the SSO/UC-related software, a CCOW-enabled and SSO/UC-aware RPC Broker component (i.e., TCCOWRPCBroker, a non-visual component) was created to provide single sign-on capability. Thus, when a VistA CCOW-enabled application, such as Computerized Patient Record System (CPRS), is recompiled with the TCCOWRPCBroker component and other required code modifications are made, that application would then become SSO/UC-aware and capable of single sign-on (SSO).

ℹ **REF:** For more detailed information on the application developer procedures and code modifications needed to make CCOW-enabled RPC Broker-based applications SSO/UC-aware, please refer to the "RPC Broker-based Client/Server Applications" topic in Chapter 4, "Making VistA Applications SSO/UC-aware," in this manual.

The RPC Broker SSO/UC functionality was introduced with RPC Broker Patches XWB*1.1*35 (server-side) and XWB*1.1*40 (client-side).

ℹ **REF:** For more information on the RPC Broker, please refer to the RPC Broker documentation located at the following Web address:

http://vaww.vista.med.va.gov/vdl/Infrastructure.asp#App23

VistALink

VistALink connects Java-based rich client applications running on a Microsoft Windows client workstation to a VistA M Server. This connection allows the client application to execute remote procedure calls (RPCs) on the VistA M Server. Like the RPC Broker, it uses the Access and Verify codes to authenticate a user to VistA.

For the SSO/UC-related software, CCOW-enabled and SSO/UC-aware VistALink classes were created to provide single sign-on capability. Thus, when a VistA CCOW-enabled application, such as Care Management (CM), is recompiled with the CCOW-enabled VistALink classes and other required code modifications are made, that application would then be SSO/UC-aware and capable of single sign-on (SSO).

NOTE: VistALink has incorporated CCOW functionality in VistALink V. 1.5.

REF: For more detailed information on the application developer procedures and code modifications needed to make CCOW-enabled VistALink-based applications SSO/UC-aware, please refer to the "VistALink-based Client/Server Applications" topic in Chapter 4, "Making VistA Applications SSO/UC-aware," in this manual.

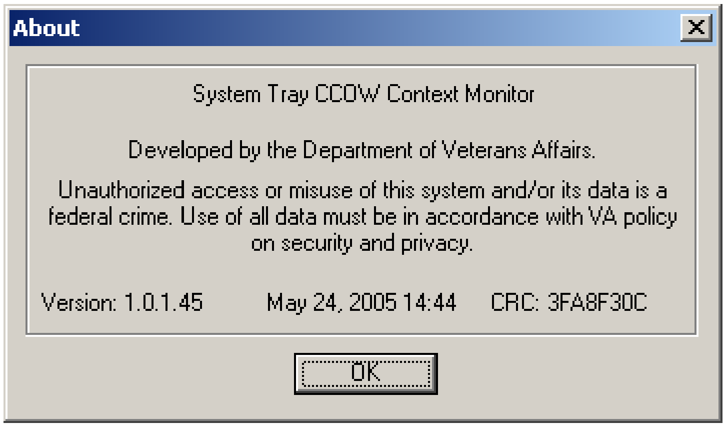
The VistALink SSO/UC functionality is introduced in VistALink V. 1.5.

REF: For more information on VistALink, please refer to the Application Modernization Foundations Web site located at the following Web address:

http://vaww.vista.med.va.gov/migration/foundations/vl/index.htm

CCOW Context Monitor

The CCOW Context Monitor V. 1.0 is a CCOW-enabled, and User Context (UC)-aware (small) rich client application that can be used to monitor and display the name of the current user in context:



System Tray CCOW Context Monitor

Developed by the Department of Veterans Affairs.

Unauthorized access or misuse of this system and/or its data is a federal crime. Use of all data must be in accordance with VA policy on security and privacy.

Version: 1.0.1.45 May 24, 2005 14:44 CRC: 3FA8F30C

Figure 2-2. CCOW Context Monitor About Box (sample test version)

NOTE: The ISS SSO/UC Development Team developed the CCOW Context Monitor application software as an additional monitoring tool and is *not* required by the SSO/UC-related software and its functionality.

REF: The latest test version of the CCOW Context Monitor application is available for download at the following Web addresses:

http://vista.med.va.gov/kernel/sso/download.asp#all

The CCOW Context Monitor application runs in the background and is automatically started at system start up. Users know the CCOW Context Monitor is running by the icon displayed in the Windows taskbar status area (a.k.a. system tray):

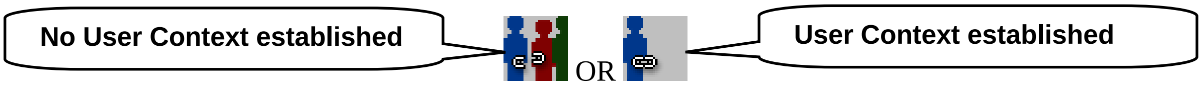


Figure 2-3. CCOW Context Monitor Icons

User Context *Not* Established

Client Workstation System Tray

When no User Context has been established, users will see the CCOW Context Monitor icon with the *multi*-user silhouette and *broken* chain link (i.e.,) displayed in the client workstation system tray, as shown below:

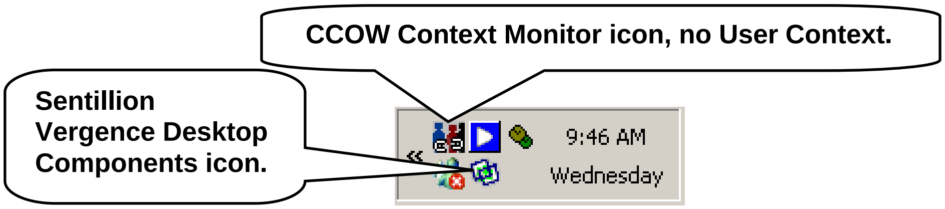

CCOW Context Monitor icon, no User Context.
Sentillion Vergence Desktop Components icon.
9:46 AM
Wednesday

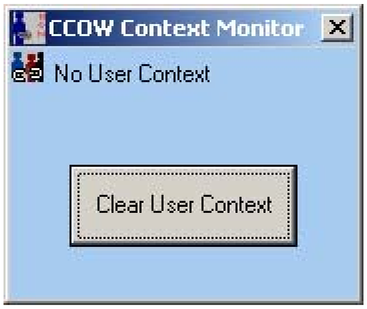
Figure 2-4. CCOW Context Monitor icon in Windows system tray—No User Context established

When users hover over the CCOW Context Monitor icon in the system tray *before* User Context has been established, they will see the following tool tip text displayed:

No User Context

Application Window

When users double click on the CCOW Context Monitor icon the in the system tray (see Figure 2-4), the CCOW Context Monitor application opens, as shown below:



No User Context

Figure 2-5. CCOW Context Monitor application window—No User Context established

In this example, User Context has *not* been established (i.e., the user signed on has not started any CCOW-enabled and SSO/UC-aware applications on this workstation), so users will see "No User Context" displayed next to the CCOW Context Monitor icon in the application window.

The CCOW Context Monitor frequently checks the current User Context for any changes and the words "Joining Context" will periodically appear above the "Clear User Context" button indicating that the system is polling the system to see if any user is in context, as shown below:

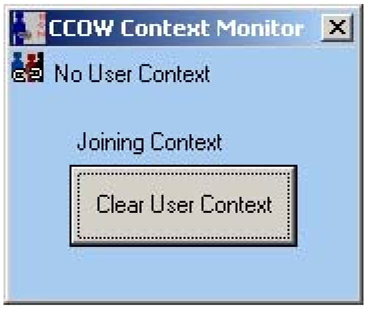


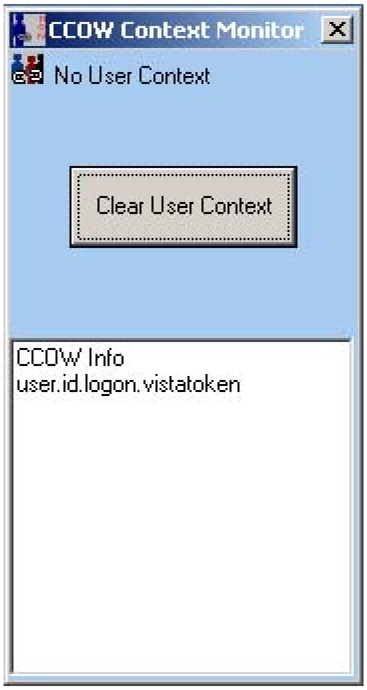
Figure 2-6. CCOW Context Monitor—Joining Context

If the User Context has changed, the CCOW Context Monitor application window will change and display the name of the user that has joined User Context (see Figure 2-9).

NOTE: Pressing the Close button (☒) simply closes the application window on the client workstation; it does *not* shut down the CCOW Context Monitor running in the background (i.e., icon remains in the client workstation system tray).

REF: For more information on the Clear User Context button, please refer to the "Clearing User Context" topic in this chapter.

When users double click on the CCOW Context Monitor icon displayed to the left of "No User Context" in the application window, they will see the following expanded window:



No User Context

CCOW Info
user.id.logon.vistatoken

Figure 2-7. CCOW Context Monitor application window (expanded)—CCOW information displayed when *no* User Context established

When users again double click on the CCOW Context Monitor icon displayed to the left of "No User Context" in the application window, the window will return to the original view (see Figure 2-5).

User Context Established

Client Workstation System Tray

Once a User Context has been established (i.e., a user has signed onto at least one CCOW-enabled and SSO/UC-aware application), users will see the CCOW Context Monitor icon with the *single* user silhouette and an *unbroken* chain link (i.e.,) displayed in the client workstation system tray, as shown below:

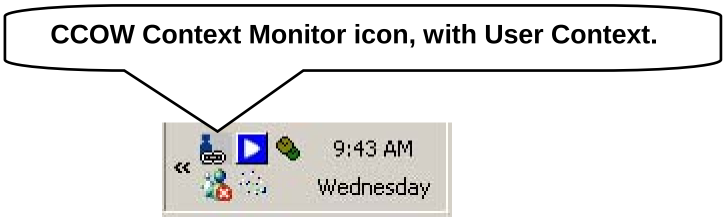
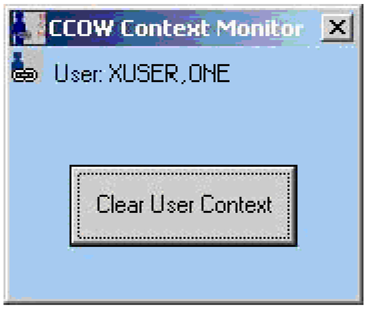


Figure 2-8. CCOW Context Monitor icon in Windows system tray—User Context established

When users hover over the CCOW Context Monitor icon in the system tray *after* User Context has been established, they will see the name of the current user in context (e.g., XUSER,ONE, see Figure 2-9). This is the user that initiated the first CCOW-enabled and SSO/UC-aware application on this workstation.

Application Window

When users double click on the CCOW Context Monitor icon in the system tray (see Figure 2-8), the CCOW Context Monitor application opens, as shown below:



User: XUSER,ONE

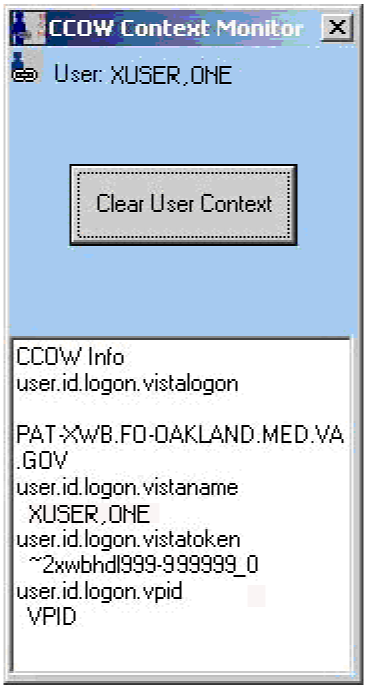
Figure 2-9. CCOW CM open window—User Context established

In this example, User Context has been established, so users will see the name of the user in context (e.g., "User: XUSER,ONE") displayed next to the CCOW Context Monitor icon in the application window.

NOTE: Pressing the Close button (☒) simply closes the application window on the client workstation; it does *not* shut down the CCOW Context Monitor running in the background (i.e., icon remains in the client workstation system tray).

REF: For more information on the Clear User Context button, please refer to the "Clearing User Context" topic in this chapter.

When users double click on the CCOW Context Monitor icon displayed to the left of the user name in context (e.g., "User: XUSER,ONE") in the application window, they will see the following expanded window:



CCOW Info
user.id.logon.vistalogon

PAT-XWB.FO-OAKLAND.MED.VA
.GOV
user.id.logon.vistaname
 XUSER,ONE
user.id.logon.vistatoken
 ~2xwbhdl999-999999_0
user.id.logon.vpid
 VPID

Figure 2-10. CCOW Context Monitor open window (expanded)—CCOW information displayed when User Context established

This is the expanded view of the CCOW Context Monitor application window. This view provides more detailed system CCOW data for the user that signed onto a CCOW-enabled and SSO/UC-aware application. It can be useful for troubleshooting purposes (e.g., IRM personnel).

When users again double click on the CCOW Context Monitor icon displayed to the left of the user name in context (e.g., "User: XUSER,ONE") in the application window, the window will return to the original view (see Figure 2-9).

CCOW Context Monitor Menu Options

By right clicking (left clicking for left-handed users) on the CCOW Context Monitor icons in the system tray (see Figure 2-3), users will see the current user in context and the available menu options, as shown below:

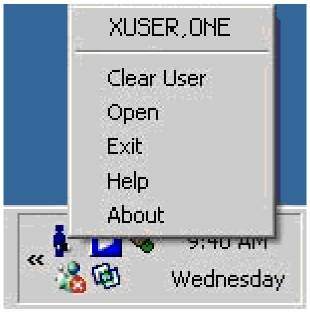


Figure 2-11. CCOW Context Monitor—Menu options

The first item displayed will be the name of the current user in context, if any. In this example, the user name in context is "XUSER,ONE." If there is no user in context, "No User Context" is displayed on top of the menu options.

The CCOW Context Monitor menu options include the following commands:

- **Clear User—**This command is used to clear the User Context (i.e., set User Context to null). Upon clearing the User Context, as a rule, all CCOW-enabled and SSO/UC-aware applications (e.g., Care Management and CPRS) should automatically shut down/close.

  REF: For more information on the Clear User command, please refer to the "Clearing User Context" topic in this chapter.

- **Open—**This command opens the CCOW Context Monitor application, which will display the current user name in context (see Figure 2-9). If no user is in context, users will see "No User Context" (see Figure 2-5).
- **Exit—**This command exits the menu display *and* shuts down the CCOW Context Monitor application running in the background (i.e., icon is removed from the client workstation system tray).
- **Help—**This command displays the CCOW Context Monitor help file (i.e., ContextMon.hlp).
- **About—**This command displays the CCOW Context Monitor About Box (see Figure 2-2).

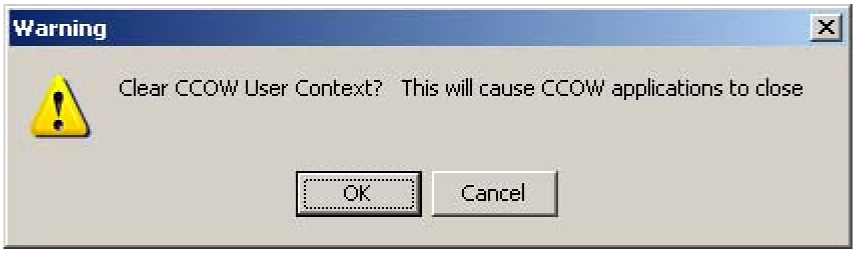
Clearing User Context

Clearing user context sets the User Context stored on the Context Vault to null. Upon clearing the User Context, as a rule, *all* CCOW-enabled and SSO/UC-aware applications (e.g., Care Management and CPRS) should automatically shut down/close.

Users can clear User Context by either of the following methods:

- Pressing the "Clear User Context" button in the CCOW Context Monitor application (see Figure 2-9).
- Using the "Clear User" command on the CCOW Context Monitor menu (see Figure 2-11).

After choosing one of these methods, users will get the following confirmation message:

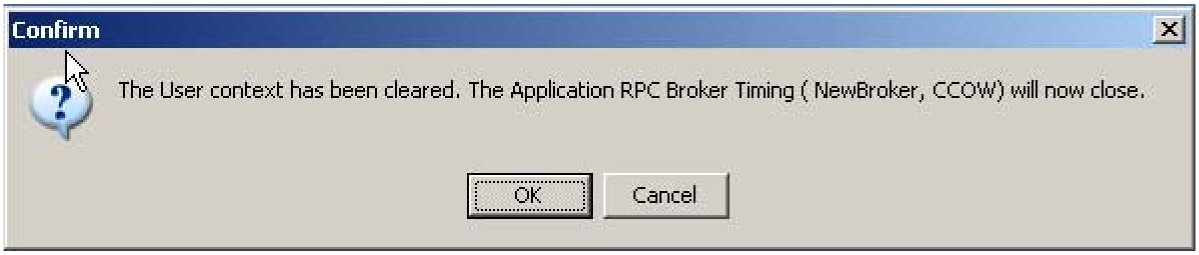


Clear CCOW User Context?   This will cause CCOW applications to close

Figure 2-12. CCOW Context Monitor notification message when clearing CCOW User Context

If the user presses the "OK" button, the User Context is set to null on the Context Vault and all CCOW-enabled and SSO/UC-aware applications running on the client workstation will shut down. Once all applications have shut down, Kernel utilities delete the Kernel CCOW login token on the VistA M Server.

After confirming that the user want to clear the User Context, some CCOW-enabled and SSO/UC-aware applications may display a follow-up notification message before shutting down, such as the following:



The User context has been cleared. The Application RPC Broker Timing (NewBroker, CCOW) will now close.

Figure 2-13. Sample application shut-down confirmation message after User Context has been cleared (This message is from the NewBrokerTimingCCOW.exe sample application)

⚠ **Any other *non*-CCOW-enabled and SSO/UC-aware VistA applications open on the workstation will continue to run and will *not* be affected when a user clears User Context. Only CCOW-enabled and SSO/UC-aware VistA applications will shut down.**

Sentillion Vergence Context Vault

The SSO/UC-related software requires that the Sentillion Vergence Context Vault COTS software (see Table 6-5) be installed on a server that is accessible to sites that will be running CCOW-enabled applications. It *must* be properly configured for User Context, including the required application names and passcodes.

The Sentillion Vergence Context Vaults stores the User Context and interfaces with the Sentillion Vergence Desktop Components loaded on every client workstation that is running CCOW-enabled applications.

NOTE: This COTS software is already purchased, being installed, and configured as part of the Clinical Context Management Project (CCOW) release for Patient Context. However, for SSO/UC, a separate User subject license is required. This license has already been purchased, but it now *must* be installed.

REF: For more information on the CCOW Package Release, please refer to the Context Management Project (CCOW) release documentation available on the EVS Anonymous Directories.

REF: For a more detailed overview of the Sentillion Vergence Context Vaults, please refer to the "Context Vault Overview" and "Context Vault Features" topics in the "Introduction" chapter in the *Sentillion Vergence Context Vault 3.3 User's Guide*.

Sentillion Vergence Desktop Components

The SSO/UC-related software requires that the Sentillion Vergence Desktop Components COTS software (see Table 6-5) be installed on every client workstation that will be running CCOW-enabled applications.

The Sentillion Vergence Desktop Components COTS software runs in the background and is automatically started at system start up. Users know the Sentillion Vergence Desktop Components are running by the icon displayed in the Windows taskbar status area (a.k.a. system tray, see Figure 2-4).

These components are used to interface CCOW-enabled applications running on client workstations with the Sentillion Vergence Context Vaults on the server where the User Context is stored.

NOTE: This COTS software is already purchased, being installed, and configured as part of the Clinical Context Management Project (CCOW) release for Patient Context.

REF: For more information on the CCOW Package Release, please refer to the Context Management Project (CCOW) release documentation available on the EVS Anonymous Directories.

REF: For a more detailed overview of the Sentillion Vergence Desktop Components, please refer to the "Vergence System Architecture Overview" topic in the "Introduction to the Vergence Desktop components" chapter in the *Sentillion Vergence Desktop Components 3.3 Installation Guide*.

Sentillion Software Development Kit (SDK)

Sentillion's Software Development Kit (SDK) COTS software is used by developers for developing CCOW-enabled applications. Two types of SDK are provided:

- COM-based Rich Client SDK—Provided as a COM component.
- Java-based Web Client SDK—Intended to CCOW-enable Web applications written in Java.

NOTE: As of this writing, Sentillion does *not* provide an SDK for Java-based rich client/server applications (i.e., VistALink or FatKAAT-based applications).

Sentillion SDK Type	Description
COM-based Rich Client SDK	This SDK is used for COM-based rich client applications, which can be used to CCOW-enable VHA's COM client applications developed in Borland Delphi that connect to the VistA M Server using the RPC Broker (e.g., Computerized Patient record System [CPRS]).
Java-based Web Client SDK	This SDK is used for Java-based web client applications, which can be used to CCOW-enable those applications. VHA OI HSD&D Health Data Systems (HDS) created Java wrapper libraries for Sentillion's Java-based Web SDK that allows them to be used by a Java-based rich client. This is done by making the rich client application simulate a Web application (entirely contained on the client) including implementing a "tiny" Web server that runs on the client. For SSO/UC, VistALink makes use of those Java wrapper libraries to support the synchronizing of the User subject in addition to the Patient subject. This has achieved a somewhat similar interface for Java applications to those of COM applications. Java applications do have a few additional responsibilities that COM-based applications do not have.

Table 2-1. Sentillion SDK descriptions

The implementations of the RPC Broker COM-based component and VistALink Java-based login classes are somewhat different. This can be attributed to the differences between the COM and Java implementations of the Sentillion CCOW SDK.

II. Developer Guide

This is the Developer Guide section of this supplemental documentation for the SSO/UC-related software. It is intended for use in conjunction with the SSO/UC Project (Iteration 1). It details the programmer-related SSO/UC documentation (e.g., developer procedures needed to make CCOW-enabled applications SSO/UC-aware, APIs exported with the SSO/UC-related software, etc.).

3. SSO/UC Installation Instructions for Developers

Preliminary Considerations: Developer Workstation Requirements

The following minimum hardware and software tools/utilities are required by developers when developing RPC Broker/Delphi-based or VistALink/Java–based rich client applications that are CCOW-enabled and SSO/UC-aware:

Minimum Hardware/Software Requirement	Description
Workstation Hardware	80x86-based client or server workstation.
Operating System	One of the following 32-bit operating systems: • Microsoft Windows XP • Microsoft Windows 2000
Development-related Software	The following development-related software is required, (or optional depending on your development environment) in order to develop RPC Broker/Delphi-based or VistALink/Java–based rich client applications that are CCOW-enabled and SSO/UC-aware: • (required) SSO/UC-related Software—SSO/UC client software, see Table 1-1 and Table 1-2. • (optional) Borland Delphi Integrated Development Environment (IDE)—Commercial-Off-The-Shelf (COTS) software for development of RPC Broker/COM-based rich client applications. • (optional) RPC Broker Development Kit (BDK) V. 1.1 (i.e., XWB*1.1*40) for development of COM-based rich client applications. • (optional) Java Software Development Kit (SDK)—COTS software for development of VistALink/Java-based rich client applications. • (optional) VistALink V. 1.5 for development of Java-based rich client applications. • (required) Sentillion Vergence Desktop Components. **REF:** For more information on installing the Sentillion Vergence Desktop Components, please refer to the Sentillion Vergence Desktop Components installation CD-ROM that contains the following: – *Desktop Installation Guide* – Microsoft's Java Virtual Machine (%windir%\system32\msjava.dll) Version

Minimum Hardware/Software Requirement	Description
	5.00.3188 or later. **REF:** For more information on configuring files and integrating SSO/UC functionality with rich client-based software applications, please refer to Chapter 4, "Making VistA Applications SSO/UC-aware," in this manual.
Citrix Environment	In a Citrix environment, the following changes are required: • NT 4, Service Pack 3—Citrix Metaframe 1.8, Service Pack 3. • Windows 2000—Citrix Metaframe XP, Feature Release 1.0.
Network Communications Software/Capability **REF:** For more information on telecommunications support, please visit the VA Office of Information and Technology (OIT) Home Page: http://vaww.va.gov/oirm/telecom/	All developer client or server 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.

Table 3-1. Developer minimum hardware and software tools/utilities required for CCOW-enabled and SSO/UC-aware application development

Dependencies—SSO/UC-related Software

The following table shows the SSO/UC-related software version dependencies with RPC Broker and VistALink software:

Software	Version	SSO/UC Client Software Release/Distribution
RPC Broker	1.1	The client-side RPC Broker Development Kit (BDK) executable (i.e., XWB1_1P40PG.EXE), as referenced in RPC Broker Patch XWB*1.1*40 (i.e., VistA M Server patch). This executable contains the TCCOWRPCBroker Component. It is available for download from the Enterprise VistA Support (EVS) Anonymous directories.
VistALink	1.5	This is a runtime and development tool providing connection and data conversion between Java and M applications in client-server and n-tier architectures. It is available for download from the Enterprise VistA Support (EVS) Anonymous directories.

Table 3-2. Dependencies—SSO/UC, RPC Broker, and VistALink software

REF: For a list of RPC Broker dependent VistA M Server patches and installation instructions, please refer to the *RPC Broker Installation Guide (Version 1.1, Patch XWB*1.1*40)*.

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

SSO/UC Installation Instructions: Virgin Installation

The following instructions are only required for those client workstations to be used by programmers to develop CCOW-enabled and SSO/UC-aware rich client-based software applications.

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

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

The following files are needed to install the SSO/UC-related Developer Client Workstation software:

File Name	Type	Description
SSO-UC_ README.TXT	ASCII	**Readme File** (manual). This file provides any pre-installation instructions, last minute changes, new instructions, and/or additional information to supplement the manuals. Read all sections of this file prior to following the installation instructions in the *Single Sign-On/User context (SSO/UC) Installation Guide* (i.e., SSO-UC_INSTALLGUIDE.PDF).
SSO-UC_ INSTALLGUIDE.PDF	Binary	**Installation Guide** (manual). Use in conjunction with the Readme text file (i.e., SSO-UC_README.TXT).
XWB1_1P40PG.EXE RPC Broker V. 1.1; see RPC Broker Patch XWB*1.1*40 on FORUM	Binary	**RPC Broker/Delphi-based Programmer Client Workstation** (client software). Available on the Enterprise VistA Support (EVS) Anonymous directories. This is a self-installing executable that contains the following: • Broker Development Kit (BDK): Provides the TCCOWRPCBroker Delphi Component for Delphi Broker development. • Broker.hlp: The complete online reference to the BDK. • BrokerProgPref.exe: Sets BDK developer preferences. • ServerList.exe: A configuration tool to edit client connections to RPC Broker servers. **NOTE:** For more information, please refer to the RPC Broker documentation set available on the VDL: http://www.va.gov/vdl/Infrastructure.asp?appID=23

File Name	Type	Description
XOB_1_5.KID VistALink V. 1.5 (J2M)	ASCII	**VistALink/Java-based Programmer Client Workstation** (server software). Available on the Enterprise VistA Support (EVS) Anonymous directories.
XOB_1_5.ZIP VistALink V. 1.5 (J2M)	Binary	**VistALink/Java-based Programmer Client Workstation** (client software). Available on the Enterprise VistA Support (EVS) Anonymous directories.

Table 3-3. Distribution files—SSO/UC-related developer client workstation files

2. Create a SSO/UC Staging Folder *(required)*

Create a SSO/UC Staging Folder on your Developer Client Workstation. This will be referred to as the **<STAGING_FOLDER>** for the rest of the instructions.

3. Review/Use SSO/UC-related Component/Classes for Client/Server-based Applications *(recommended)*

To build your VistA client/server applications that are CCOW-enabled and SSO/UC-aware, you need to configure and include either of the following component/classes depending on your development environment:

- RPC Broker/Delphi-based—TCCOWRPCBroker Component
- VistALink/Java–based—CCOW-enabled and SSO/UC-aware VistALink login classes

Each VistA/Health*e*Vet-VistA rich client-based application requiring Authentication and Authorization against Kernel on the VistA M Server will use the standard Kernel login routines.

REF: For more information on integrating SSO/UC with VistA client/server-based software applications, please refer to Chapter 4, "Making VistA Applications SSO/UC-aware," in this manual.

Congratulations! You have now completed the installation of SSO/UC-related software on the *Developer Client Workstation*

4. Making VistA Applications SSO/UC-aware

This chapter describes how application developers can modify their CCOW-enabled rich client/server VistA applications to be Single Sign-On/User Context (SSO/UC)-aware.

This chapter discusses the following topics:

- Assumptions When Implementing SSO/UC
- Application Rules for User Subject Context Changes
- Rich Client Application Procedures to Implement SSO/UC (listed by application type):
  - RPC Broker-based Client/Server Applications (i.e., COM client application developed in Borland Delphi)
  - VistALink-based Client/Server Applications (i.e., Java client application)

Assumptions When Implementing SSO/UC

The following assumptions are made regarding application developers and VistA software applications when implementing SSO/UC (Iteration 1):

- **Developer Training—**Application developer is already knowledgeable/trained in implementing CCOW (e.g., Patient Context).

  NOTE: It is recommended that if developers have no CCOW experience (e.g., CCOW-enabling an application) that they go through Sentillion's CCOW immersion training.

- **CCOW-enabled Applications—**Application is already CCOW-enabled for at least one CCOW subject (e.g., Patient Context).
- **CCOW GUI Icon—**Application has the standard CCOW GUI icon in place that allows breaking/rejoining context.
- **Context Changes—**Application is already listening for context changes (e.g., Patient Context).
- **User Context—**Application only uses User Context for login and logoff, not for any other purpose.
- **Login at Startup—**Applications automatically initiate login at application startup (i.e., users are presented with an Access/Verify login dialogue).
- **Login Mode—**Applications are *not* architected to have a between-user "logged out" mode; either the initial user is logged in, or the application shuts down.
- **Application Name and Passcodes—**An application's name and passcode is used to authenticate the application to the Sentillion Vergence Context Vault. The RPC Broker login component and VistALink login classes use their own application name and secure passcode to bind to context whenever they need to write/set secure subjects (i.e., User Context) in the Context Vault.

  Application developers and administrators do *not* need to make any changes in application names or passcodes for the RPC Broker login component and VistALink login classes. However, the

RPC Broker and VistALink application's name and passcode *must* be stored and configured on the Sentillion Vergence Context Vault. The application name and passcode *must* be configured using the Context Administrator.

REF: For questions regarding the RPC Broker/VistALink passcodes on the Sentillion Vergence Context Vault, please contact the CCOW Team at the following Web address:

http://vista.med.va.gov/ccow/

- **Client Contextor or Context Vault Available—**If the CCOW-enabled and SSO/UC-aware application is run on a client workstation that does *not* have the Contextor installed or is *not* pointed correctly at a Context Vault, the application will come back with the Contextor variable set to nil (i.e., no object).

  These actions related to CCOW are all dependent upon the use of the Contextor, and if this is *not* indicated as available or nil, then the application will attempt to sign on in a normal (non-CCOW User Context) manner. In addition, if it is the first connection the user has made, no attempt will be made to set the Kernel CCOW login token into the Context Vault.

  NOTE: The CCOW-enabled and SSO/UC-aware RPC Broker login components and VistALink login classes are designed to handle this fault condition.

- **VistA M Server is CCOW-enabled—**If the CCOW-enabled and SSO/UC-aware application is run on a client workstation that supports CCOW, but the VistA M Server is *not* CCOW-enabled (i.e., does *not* have the updated XUS routines needed to recognize the Kernel CCOW login token that is passed in), the user will be presented with a regular sign-on screen (i.e., require users to enter their Access and Verify codes).

  If the RPC to return the Kernel CCOW login token (after an initial sign-on) is *not* present, there would be an error message indicating that the RPC is not present on the system or added to the Context.

  NOTE: The CCOW-enabled and SSO/UC-aware RPC Broker login component and VistALink login classes are designed to handle this fault condition.

Application Rules for User Subject Context Changes

Assuming a VistA CCOW-enabled and SSO/UC-aware application has started up and logged in successfully, the application should watch for changes in the User Context (i.e., User subject) and respond to those changes.

The following guidelines/rules should be used for application detection and processing of User subject changes:

- **General Application Rules—**CCOW-enabled and SSO/UC-aware applications should handle the following general application rules:
  - Applications should look for any User Context values not just VA-specific tokens. This will simplify the initial implementation and ongoing maintenance in case the VA has to change the way it stores user information into CCOW in the future.
  - Changes to User Context (from one value to another) are ignored. This avoids problems with updating of VistA tokens and allows for logins to multiple VistA M Servers in the future (if/when the CCOW SSO/UC Project supports it).
- **Application Startup—**After VistA login and when rejoining the clinical context (from a broken link), CCOW-enabled and SSO/UC-aware applications should check to see if the User subject contains any values at all:
  - User Subject Exists (*not* null)**—**If the User subject has any values, set a global User Context flag in the application to True (e.g., "hadUserContextAtOneTimeInThePast"). This flag will be used during context change notification messages to determine appropriate behavior. It indicates that this application is making use of the common User Context for single sign-on.
  - User Subject is Null**—**If the User subject is empty, set the application's global User Context flag to False (e.g., "hadUserContextAtOneTimeInThePast"). This indicates that this application is *not* making use of the common User Context for single sign-on.
- **Rejoining Common Context—**When an application rejoins context, it should *not* try to set or "tune" to the User Context in any way:
  - If the User Context has since gone empty (null), the rejoining application should do nothing.
  - If the User Context has a value, the rejoining application should do nothing.
  - If the user value in the context is different than the user for the rejoining application, the rejoining application should do nothing.
- **Pending Phase of Context Change—**In the pending phase of a context change (e.g., calls to ContextChangePending), CCOW-enabled and SSO/UC-aware applications should respond with an appropriate complaint if either of the following conditions exist:
  - Changing the Patient Context would cause the user to lose data.
  - Shutting the application down would cause the user to lose data.

- **Commit Phase of Context Change —**In the commit phase of a context change (e.g., calls to ContextChangesCommitted), the CCOW-enabled and SSO/UC-aware applications should check to see if the User subject contains any values:
  - User Subject Exists (*not* null)**—**If User subject has any values, set the application's global User Context flag to True.
  - User Subject is Null**—**If User subject is empty and the application's global User Context flag is True, applications *must* shut down.

    Application developers should perform any necessary/appropriate housekeeping chores to properly/safely shut down the application when User Context changes to null.

Rich Client Application Procedures to Implement SSO/UC

RPC Broker-based Client/Server Applications

All VistA application developers with RPC Broker-based applications (i.e., COM client applications developed in Borland Delphi) need to perform the following procedures in order to make their CCOW-enabled applications SSO/UC-aware:

Assumptions for RPC Broker-based Applications:

- The client application is Borland Delphi-based.
- The application has already been CCOW-enabled for at least one CCOW subject.
- The CCOW-enabled client application has already implemented the following:
  - A context observer/participant class.
  - Graceful shutdown of CCOW component when the application closes.
  - Proper visual display of the interactive CCOW GUI icon reflecting and controlling the current joined/broken status of the CCOW links.

1. Replace TRPCBroker with TCCOWRPCBroker Component *(required)*

a. Open Delphi and load your application.

   The CCOW-enabled RPC Broker is a TCCOWRPCBroker component that is derived from TRPCBroker. An application can be converted to run with the TCCOWRPCBroker.

b. Developers should note the name of the current TRPCBroker component. If developers do not have the component on the form, then they should locate where it is declared and created and note its name.

c. Remove the TRPCBroker component from the form and replace it with the TCCOWRPCBroker component. Change the name of the component from the default "TCCOWRPCBroker1" to the original Broker name. If developers do not have the component on the form, they should change the declaration of their instance from TRPCBroker to TCCOWRPCBroker, and add CCOWRPCBroker to the uses clause.

2. Set New Contextor Property After Creating Contextor Instance *(required)*

Developers should locate the code where they set the Connected property to True, first set a context, or make a remote procedure call (i.e., where they start interacting with the VistA M Server). Developers *must* create an instance of the TContextorControl and initialize it with the Run command prior to this code.

To activate the User Context functionality, set the Contextor property (i.e., public, run-time property) of the TCCOWRPCBroker to the value of the Contextor object (i.e., instance of the

TContextorControl that the application is using for its context) prior to setting the Connected property to True.

NOTE: If the Client Contextor or Context Vault is *not* available, the Contextor variable will be nil**.** Therefore, developers would *not* set the Contextor property of the TCCOWRPCBroker component. Thus, a normal (non-CCOW User Context) sign-on will be attempted. Developers only set the Contextor property of the TCCOWRPCBroker component if the Contextor is *not* nil (the default value is nil).

The RPC Broker needs to know the internal instance of the ContextorControl. It uses this Contextor to see if the application has a User Context and Kernel CCOW login token to use for login.

NOTE: For multi-division users, the RPC Broker does *not* request the division, since the VistA M Server code is already setting the division to the division selected when the Kernel CCOW login token was created (i.e., the division associated with the token that was generated for the Context Vault).

3. Call Contextor Run Method Before Connecting with RPC Broker *(required)*

The application *must* not only create the application's instance of TContextorControl, but *must* also use the Run method prior to setting the Connected property of TCCOWRPCBroker to True. The application name *must* be provided and *must* be an entry in the Context Vault. If the application name has a #-suffix multiple, instances of the same application will be permitted to join the context. For example:

```
ContextorControl1.Run('CPRSChart#', '', TRUE, '*')
```

If the application does not have the #-suffix, only a single instance of the application will be permitted to join the context. For example:

```
ContextorControl1.Run('CPRSChart', "", True, '*')
```

4. Implement Application Rules for User Subject Context Changes *(required)*

Upon notification of a User Context change application developers *must* code their CCOW-enabled applications to do the following:

a. Determine if subject is a User Context change.

The TCCOWRPCBroker has six read-only properties that the application can check related to the data in the Context Vault which is related to the User Context.

- CCOWLogonIDName (String)—Contains the value: user.id.logon.vistalogon
- CCOWLogonIDValue (String)—Contains the domain name of the institution the user logged into (e.g., PAT-XWB.FO-OAKLAND.MED.VA.GOV).
- CCOWLogonName (String)—Contains the value: user.id.logon.vistaname

- CCOWLogonNameValue (String)—Contains the contents of the NAME field (#.01) from the NEW PERSON file (#200) for the user (e.g., XUSER,ONE).
- CCOWLogonVistAToken (String)—Contains the value: user.id.logon.vistatoken
- CCOWLogonVPID (String)—Contains the value: user.id.logon.vpid

NOTE: The Kernel CCOW login token will also contain the Department of Veterans Affairs Personal Identification (VPID), which will be null until the VA enterprise use of VPID is enacted. The VPID will be retrieved from the NEW PERSON file (#200) , where it will be stored in addition to being stored in national directories. There will also be a VPID entry stored in the Sentillion Vergence Context Vault.

b. When checking the events associated with changes in context, the application *must* remember to check for changes associated with the User Context in addition to any others that may be active. The only action that is expected from an application for a User Context is that it should terminate if the User Context is not defined when it was defined previously. There are two methods with the TCCOWRPCBroker component that can be used in combination on the OnCommitted event to determine if this is the case:

- IsUserCleared method—Use this method to determine whether or not the User Context value has changed to null.
- WasUserDefined method—Use this method to determine whether or not the User Context value was ever defined

NOTE: It is recommended that applications check the context data during the OnPending event and set any flags as needed because the context *cannot* be changed between the OnPending and the OnCommit event notifications.

For example, using the IsUserCleared method, if a User Context value has been established, it will return a value of False. Subsequently, if the value returned during an OnPending event was False, and the value returned during the subsequent OnCommit event is True, then User Context has been cleared and the application should shut down.

```
function IsUserCleared: Boolean;
```

REF: For a list of application rules, please refer to the "Application Rules for User Subject Context Changes" topic in this chapter.

The following is an example of code used by an application to initialize the ContextorControl and make the connection via the CCOW-enabled TCCOWRPCBroker component.

```
procedure TfrmCanvas.DoConnect;
var
  TokenVal: String;
  userPrivilege: AccessPrivilege;
  ChangeCursor: Boolean;
begin
    if (NContext = 0) and (Host.BrokerPort <> '') then
    begin
      StatusBar1.SimpleText := 'Setting Single Sign-On User Context';
      if Screen.Cursor = crDefault then
        ChangeCursor := True
      else
        ChangeCursor := False;
      if ChangeCursor then
        Screen.Cursor := crHourGlass;
  try
    // Join the common context on startup. To write secure subject(s),
    // provide a passcode with the application name. The application name
    // and passcode must be configured using the Context Administrator in
    // order for this application to successfully set secure
    // subjects.

    // Start the Contextor which will join the context. Indicate desire
    // to be surveyed and notified of all subjects. Append '#' to the app
    // name to permit multiple instances of this app to participate in the
    // context.
    ContextorControl.Run( APP_NAME + '#'
                   , APP_PASSCODE1 + APP_PASSCODE2
                   , TRUE  // VARIANT_TRUE == -1 in COM
                   , '*');

    // Upon startup, determine if there is a context already set that this
    // app should tune to (e.g., an existing user selected in another CCOW
    // app, etc).
    // If not, go to a resting display (no user data shown).
    CheckContextAndDisplay();
    RPCB.Contextor := ContextorControl;
  except
    on exc : EOleException do
    begin
      ShowOleException(exc);

      // If a Context Manager error occurred, app should run non-CCOW
      // enabled.
      if (ITF_E_CONTEXT_MANAGER_ERROR = exc.ErrorCode) then
      begin
        // This sample displays a message and terminates. A normal app
        // should set its state to run non-CCOW-enabled.
        ShowMessage('Problem starting the Context Manager');
        Application.Terminate();
      end;
    end;
  end;
  RPCB.Connected := True;
  if not RPCB.Connected then
  begin
    ShowMessage('Connection Failed');
    Application.Terminate();
  end;
  if ChangeCursor then
    Screen.Cursor := crDefault;
  TokenVal := GetToken();
  NContext := 1;
  end;
end;
```

Figure 4-1. Sample RPC Broker code initializing ContextorControl and making connection via the TCCOWRPCBroker component

```
Tform1.OnCCOWCommit(Sender: TObject);
Begin
  // do any processing related to other contexts
  // such as Patient Context
  //
  // now check for User Context change
  //- going from defined to null or cleared
  with RPCBroker1 do
  begin
    // check if User Context was previously defined and is now cleared
    if IsUserCleared and WasUserDefined then
      Halt;  // do any processing necessary to close and halt.
  end;    // with
end;
```

Figure 4-2. Sample code using the IsUserCleared and WasUserDefined methods during OnCommitted event

5. Recompile and Test CCOW-enabled and SSO/UC-aware RPC Broker-based Application *(required)*

Developers *must* recompile their application when all edits/updates are completed.

Developers should test their application to see if it is now CCOW-enabled and SSO/UC-aware. In particular, you should test that your application can share User Context/SSO with other CCOW-enabled, User-subject-enabled VistA applications. Your application and the other VistA applications should both be capable of either establishing the User Context (first application to log in) or log in with SSO via User Context already established by another, earlier launched VistA application.

REF: For a list of application rules, please refer to the "Application Rules for User Subject Context Changes" topic in this chapter.

VistALink-based Client/Server Applications

All VistA application developers with VistALink-based applications (i.e., Java client applications) need to perform the following procedures in order to make their CCOW-enabled applications SSO/UC-aware:

Assumptions for VistALink-based Applications:

- The client application is Java-based.
- The application has already been CCOW-enabled for at least one CCOW subject, using the Java wrapper libraries for the Sentillion Web Software Development Kit (SDK), which are provided in the following packages:
  - gov.va.med.hds.cd.ccow
  - gov.va.med.hds.cd.ccow.internal
  - gov.va.med.hds.cd.ccow.ui
- The CCOW-enabled client application has already implemented the following:
  - A context observer/participant class.
  - Graceful shutdown of CCOW classes when the application closes.
  - Proper visual display of the interactive CCOW GUI icon reflecting and controlling the current joined/broken status of the CCOW links.

Examine CCOW-Enabled VistALink Sample Application *(recommended)*

A sample VistALink application that is CCOW-enabled for User Context, VistaLinkSwingSimpleCCOW, in the gov.va.med.foundations.samples package, is provided in the samples folder of the patched VistALink developer distribution. This sample application provides an example of implementing CCOW User Context and using the CCOW-enabled and SSO/UC-aware VistALink login classes.

NOTE: This sample application is CCOW-enabled for the User subject only.

REF: To take advantage of Single Sign-On (SSO) functionality, additional configuration of your CCOW Context Vault and Kernel system is required. Please refer to the *Single Sign-On/User Context (SSO/UC) Installation Guide* for more information on how to appropriately configure the following:

- Kernel on the VistA M Server
- Client workstation (with Sentillion Vergence Desktop Components)
- Sentillion Vergence Context Vault

NOTE: This sample application is CCOW-enabled for the User subject only.

NOTE: The instructions that follow assume that the required configuration to the CCOW Context Vault and Kernel system has already taken place.

To run the sample application, do the following:

a. Install the Sentillion Vergence Desktop Components on the client workstation upon which you are running the sample application.

b. Obtain the following two Java libraries from the Sentillion Web SDK:

   - WebJContextor
   - WebJDesktop

c. Obtain the following four Java libraries from the Health Data Systems (HDS) Development Team or from the Care Management software distribution. These libraries provide a Java wrapper around the Sentillion Web components.

   - ccow.core
   - ccow.ui
   - ccow
   - util

d. Follow the existing VistALink V. 1.5 instructions to set up the jaas.config and log4j configuration files for your system.

e. Follow the existing VistALink V. 1.5 instructions to set up the standard Java classpath entries needed for VistALink V. 1.5, and other settings, in the setVistaLinkEnvironment.bat batch file found in the samples folder of the Zip distribution.

f. Edit the runSwingSimpleCcow.bat batch file found in the samples folder of the Zip distribution. Update the six classpath entries with the location and names of the six Java libraries you placed on your system in Steps #b and #c above.

g. The sample application does not need to be entered as an application in the Sentillion Vergence Context Vault. However, if needed, the CCOW application name is:

   VistaLinkSwingSimpleCcow

h. Run the runSwingSimpleCcow batch file from your system's command line.

Steps to Enable User Context SSO for a Java Rich client

1. Instantiate CCOW Objects Before Creating Login Code *(required)*

Prior to instantiating CallbackHandlerSwingCCOW, the application should instantiate the following CCOW objects before invoking VistALink JAAS login code:

- ContextModule (gov.va.med.hds.cd.ccow.internal)
- SampleAppContextParticipant (an application-provided class that implements IContextObserver, IContextParticipant)

  REF: For more details on implementing a context participant, please refer to Step "5. Implement Context Participant, Using Application Rules for User Context Subject Changes *(required)*" below.

- IClinicalContextBroker (from context module's getBroker method)

Sample global declarations:

```
// ccow constants
private static final String CCOW_APPLICATION_NAME =
  "VistaLinkSwingSimpleCcow";
// not a secure binding in the app
private static final String CCOW_APPLICATION_PASSCODE = "";

// ccow object declarations
private IClinicalContextBroker ccowContextBroker;
private ContextModule ccowContextModule;
private SampleAppContextParticipant sampleAppContextParticipant;
```

Figure 4-3. Sample VistALink global declarations

Sample application constructor code:

```
// create ccow context module -- BEFORE creating the link icon and
// other visual components
ccowContextModule = new ContextModule(CCOW_APPLICATION_NAME,
  CCOW_APPLICATION_PASSCODE);

// create context participant
sampleAppContextParticipant = new SampleAppContextParticipant();

// create context broker
if (ccowContextModule != null) {
  ccowContextBroker = ccowContextModule.getBroker(this,
    sampleAppContextParticipant);
}
```

Figure 4-4. Sample VistALink application constructor code

NOTE: If the Client Contextor or Context Vault is *not* available, VistALink passes the context broker and context module to the CallbackHandlerSwingCCOW.

2. Use CCOW-enabled JAAS Callback Handler for Login *(required)*

To log in through CCOW-enabled VistALink, instead of using the normal VistALink Swing JAAS callback handler, the application uses the special CCOW-enabled VistALink JAAS callback handler, CallbackHandlerSwingCCOW, instead.

This class's constructor takes two parameters:

- CCOW context module
- CCOW context broker

3. Follow Normal JAAS Login Steps *(required)*

The following example (Figure 4-5) shows an application's login method, calling the VistALink login classes to make a connection and get a user principal back. This example also uses a separate thread to create the application context broker, which can take a few seconds, so as not to freeze the application window.

```
/**
 * Do the login
 */
private void login() {

  if (userPrincipal != null) {
    statusLabel.setText(STATUS_LABEL_CONNECTED_TEXT);
  } else {

    try {

      // create the callback handler
      CallbackHandlerSwingCCOW cbhSwing =
        new CallbackHandlerSwingCCOW(this.topFrame, this.ccowContextModule,
          this.ccowContextBroker);

      // create the LoginContext
      loginContext = new LoginContext(jaasConfigName, cbhSwing);

      // login to server
      loginContext.login();

      // get principal
      userPrincipal = VistaKernelPrincipalImpl.getKernelPrincipal(
        loginContext.getSubject());

      // only necessary if we are connected
      if ((ccowContextBroker != null) &&
     (ccowContextBroker.isConnected())) {
        // Store user context state, based on VHA user context values
        boolean hadUserContext =
          CallbackHandlerSwingCCOW.hasNonNullUserContext(
            ccowContextBroker.getContextItems());
        sampleAppContextParticipant.setHadUserContextAtOneTimeInThePast(
          hadUserContext);
      }

    } catch (Exception e) {
      JOptionPane.showMessageDialog(topFrame, e.getMessage(),
        "Login error", JOptionPane.ERROR_MESSAGE);
      statusLabel.setText(STATUS_LABEL_DISCONNECTED_TEXT);
      logout(-1);
    }
  }
}
```

Figure 4-5. Sample VistALink login classes

4. Store User Context State After Login *(required)*

The CallbackHandlerSwingCCOW class provides a static method—hasNonNullUserContext()—to return whether or not the current User Context has a VHA User Context. Using this method, your application can store the User Context state:

- True—If there is a User Context.
- False—If there is *no* User Context.

As you can see from the following code snippet taken from the login code in Step #3, it uses the hasNonNullUserContext() method and stores the result in a property maintained by the application's context participant:

```
// only necessary if we are connected
if ((ccowContextBroker != null) && (ccowContextBroker.isConnected())) {
  // Store user context state, based on VHA user context values
  boolean hadUserContext = CallbackHandlerSwingCCOW.hasNonNullUserContext(
    ccowContextBroker.getContextItems());
  sampleAppContextParticipant.setHadUserContextAtOneTimeInThePast(
     hadUserContext);
}
```

Figure 4-6. Storing User Context state after login

REF: For more details on implementing a context participant, please refer to Step "5. Implement Context Participant, Using Application Rules for User Context Subject Changes *(required)*" below.

5. Implement Context Participant, Using Application Rules for User Context Subject Changes *(required)*

For Java applications, the rules for processing context changes for the User subject should be implemented in the application class that implements IContextParticipant and IContextObserver.

The following example (Figure 4-7) shows one approach to implement the application rules (see the "Application Rules for User Subject Context Changes" topic) for processing User Context subject changes, in a context observer/participant class:

```
/**
 * This class implements the sample application's implementation of
 * following context. The sample application does not follow patient
 * context, only user context. The class implements the required interface
 * for the HDS CCOW libraries for a context observer/participant. Each
 * application should implements a class with these interfaces, specific to
 * its own app-specific needs. An instantiated object of this class is then
 * passed as a parameter by the app when creating IClinicalContextBroker
 * object.
 */
private class SampleAppContextParticipant implements IContextObserver,
 IContextParticipant {

  // used for deciding whether to sign off based on user context
  private boolean hadUserContextAtOneTimeInThePast = false;
  // object for synchronization
  private Object syncObject = new Object();

  /**
   * This method is called by the context management broker to alert the
   * component to a change in context.
   */
  public void contextChanged() {
    logger.debug("entered contextChanged.");
    logger.debug("ccowContextBroker.isConnected(): " +
      ccowContextBroker.isConnected());

    if (!ccowContextBroker.isConnected()) {
      // if we're not connected, then we're breaking context.
      this.setHadUserContextAtOneTimeInThePast(false);
    } else {
       // if we're connected, we're either 1) already in context,
       //  or 2) joining context.
      Map currentContext = ccowContextBroker.getContextItems();
      if (CallbackHandlerSwingCCOW.hasNonNullUserContext(currentContext)) {
        // set user context flag to true since we have user
        //  context after this change
        this.setHadUserContextAtOneTimeInThePast(true);
      } else if (getHadUserContextAtOneTimeInThePast()) {
        // shut down if a) already in context, b) believe
        //  user context is null and is changing from not null
        logout(0);
      } else {
        // set user context flag to false since we don't have
        //  user context after this change.
        this.setHadUserContextAtOneTimeInThePast(false);
      }
    }
  }

  /**
   * The HDS CCOW libraries call this method to inform a context
   * participant that an external context change operation has been
   * requested.
   *
   * The participant is not allowed to display any dialogs to the user
   * at this time.
   * @param contextItems a list of items in the context that could
   * potentially change
   * @return a string describing the consequences of changing the context
```

```
     * (for example, loss of unsaved data). If the proposed context changes
     * would not have adverse consequences, this method should return null.
     */
    public String reviewContextChange(Map contextItems) {
      logger.debug("entered reviewContextChange.");
      logger.debug("VHA user context items in proposed change:");
      outputVhaUserContextItems(contextItems);
      // this simple sample app never objects to a context change
      String returnVal = null;
      return returnVal;
    }

    /**
     * The HDS CCOW libraries call this method to inform a context
     * participant that an internal context change operation is about to
     * begin.
     *
     * If this participant needs the user's permission to perform any
     * action in preparation for the context change, the participant is
     * allowed to display modal dialog boxes to the user asking for
     * that permission. For example, an application might display a list
     * of unsigned items and ask the user if they want to sign the items.
     * These dialog boxes should include a Cancel button.
     *
     * @param contextItems a list of items in the context that could
     * potentially change
     * @return false if the user selected a Cancel option, true otherwise
     */
    public boolean prepareForContextChange(Set contextItems) {
      logger.debug("entered prepareForContextChange.");

      Iterator it = contextItems.iterator();
      while (it.hasNext()) {
        logger.debug("Context change item names: " +
         (IContextItemName) it.next());
      }
      // this simple sample app never objects to a context change
      boolean returnVal = true;
      return returnVal;
    }

    /**
     * Used by the application after login, to store the context item state
     * post-login.
     * @param postLoginContextItems
     */
    public void setHadUserContextAtOneTimeInThePast(boolean
      hadContextInPast) {
      // we synchronize since two threads could write/access the
      //   postLoginContextItems object
      synchronized (this.syncObject) {
        this.hadUserContextAtOneTimeInThePast = hadContextInPast;
      }
      logger.debug("Set hadUserContextAtOneTimeInThePast to " +
        hadContextInPast);
    }

    /**
     * Use getter to always get the synchronization.
     * @return current value of this property
     */
    private boolean getHadUserContextAtOneTimeInThePast() {
      synchronized (this.syncObject) {
        return this.hadUserContextAtOneTimeInThePast;
      }
    }
```

```
  /**
   * Required method; if this was clin. desktop, this method would be
   * called by the context management plugin to find out which patient
   * the context observer is displaying data for.
   *
   * @returns a map containing one or more name/value pairs that
   * correspond to data elements in the CCOW patient subject.
   */
  public Map getDisplayedPatient() {
    logger.debug("entered getDisplayedPatient.");
    // this sample app is not tracking patient
    return null;
  }

  /**
   * Output VHA user context items to debug
   * @param contextItems context items to output
   */
  private void outputVhaUserContextItems(Map contextItems) {
    // output to debug, each of the items in the context
    for (int i = 0; i <
     CallbackHandlerSwingCCOW.VHA_CCOW_USER_CONTEXT_KEYS.length; i++) {
      IContextItemName key =
        ContextItemNameFactory.getName(
          CallbackHandlerSwingCCOW.VHA_CCOW_USER_CONTEXT_KEYS[i]);
      String value = (String) contextItems.get(key);
      logger.debug("context item key '" + key + "', value: '" + value +
        "'.");
    }
  }
}
```

Figure 4-7. Sample VistALink implementation of application rules to process User Context subject changes

6. Be Sure to Shut Down CCOW Connections at Application Logout *(required)*

Because Java applications use the Sentillion Web SDK to interact with the Context Vault, it is important that when an application exits, it explicitly shuts down its CCOW connection. Otherwise, since the Web SDK is designed for operation on the often-disconnected Web, the CCOW connection will remain open and the active count of applications joined in context will *not* be decremented, which means that the context could remain open even though all applications may have exited.

For example:

```
private void logout(int exitCode) {
  // Kernel logout
  if (this.userPrincipal != null) {
  try {
    loginContext.logout();
  } catch (LoginException e) {
    // …
  }
  // always shut down CCOW before exiting
  if (ccowContextBroker != null) {
    if (ccowContextBroker.isConnected()) {
      ccowContextBroker.breakExternalLink();
    }
  }
  if (ccowContextModule != null) {
    ccowContextModule.shutdown();
  }
  // terminate the application
  System.exit(exitCode);
}
```

Figure 4-8. Sample shutdown code

7. Recompile and Test CCOW-enabled and SSO/UC-aware VistALink-based Application *(required)*

Developers *must* recompile their application when all edits/updates are completed.

Developers should test their application to see if it is now CCOW-enabled and SSO/UC-aware. In particular, you should test that your application can share User Context/SSO with other CCOW-enabled, User-subject-enabled VistA applications. Your application and the other VistA applications should both be capable of either establishing the User Context (first application to log in) or log in with SSO via User Context already established by another, earlier launched VistA application.

REF: For a list of application rules, please refer to the "Application Rules for User Subject Context Changes" topic in this chapter.

5. Application Program Interfaces (APIs) and Attributes

The Application Program Interfaces (APIs) and attributes associated with SSO/UC-related software are listed by sub-system below.

Kernel

The PROD^XUPROD API is the only public SSO/UC Application Program Interface (API) in the Kernel VistA M Server software; it is described below. All other Kernel SSO/UC-related APIs are used internally by the Kernel server-side login module.

NOTE: The PROD^XUPROD API (i.e., released with Kernel Patch XU*8.0*284) is not officially part of the SSO/UC Project (Iteration 1). However, the need for the API arose in discussions with developers during the SSO/UC Project (Iteration 1). Thus, it is being documented here.

PROD^XUPROD(): Production Vs. Test Account API

Reference Type Supported

Category Sign-on Security

IA # 4440

Description This API is called by applications to check and see if the application is running in a Production or a Test account.

The Ask if Production Account option [XU SID ASK] on the Kernel Management Menu [XUKERNEL], asks if the current account is the Production account. It returns the following values:

- True (1 or non-zero)—If the answer is "YES," the account is the Production account, so the current system ID (SID) is set as the Production SID.
- False (zero)—If the answer is "NO," the account is *not* the Production account, so a fake value is stored.

The Startup PROD check option [XU SID STARTUP] can be scheduled for startup so that when TaskMan starts the SID is checked. The first check each day gets the current SID and compares it with the stored SID to see if they match.

Format `PROD^XUPROD([force])`

Input Parameters force: (optional) The parameter value of 1 allows an application to force a full test.

Output returns: Returns a Boolean value:

True (1 or non-zero)—Production account, current SID is set as the Production SID.

False (zero)—Test account.

Example

```
>IF $$PROD^XUPROD WRITE !,"This is the production account."
```

RPC Broker

There are no public SSO/UC-related Application Program Interfaces (APIs) in the RPC Broker VistA client/server software. All RPC Broker SSO/UC-related APIs are used internally by the RPC Broker client/server login component.

NOTE: Developers should use the IsUserCleared method to determine whether or not the User Context has changed to null.

REF: For more information on this method, please refer to Step #4b in the "RPC Broker-based Client/Server Applications" topic in Chapter 3 in this manual.

VistALink

The VistALink Application Program Interfaces (APIs) are all contained within the CallbackHandlerSwingCCOW class. This topic will describe the following:

- CallbackHandlerSwingCCOW class
- Constructor Details
- Field Details
- Method Details

CallbackHandlerSwingCCOW Class

```
java.lang.Object
  └gov.va.med.foundations.security.vistalink.CallbackHandlerBase
      └gov.va.med.foundations.security.vistalink.CallbackHandlerSwing
          └gov.va.med.foundations.security.vistalink.CallbackHandlerSwingCCOW
```

All Implemented Interfaces:

javax.security.auth.callback.CallbackHandler

public class **CallbackHandlerSwingCCOW**
extends CallbackHandlerSwing

Implements the JAAS CallbackHandler interface. Use with the VistaLoginModule to invoke a Swing-based interactive logon, using the CCOW-enabled features of the VistALink login module. If user authentication is required (if a valid User Context does not exist that can be leveraged for single sign-on), input values (Access code, Verify code, Division selection, and other "user input") are collected via a set of Swing GUI dialogues by this callback handler.

To login:

1. Create a CCOW context module and broker. It *must* be securely bound to the context with a secure application passcode.
2. Create an instance of CallbackHandlerSwing, passing the Frame window parent, the context module, and broker.
3. Create the JAAS LoginContext instance, passing the instance of the callback handler as one of the parameters.
4. Invoke the JAAS login context's login method. The callback handler will invoke Swing dialogs to collect user input wherever required for login.

Constructor Detail

Constructor CallbackHandlerSwingCCOW

Description Creates a callback handler for VistALink logins, using a SWING interface, and using the CCOW-enabled features of VistALink to provide a CCOW-enabled login.

Format

```
public CallbackHandlerSwingCCOW(java.awt.Frame windowParent,
                                IContextModule applicationCcowContextModule,
                                IClinicalContextBroker applicationCcowContextBroker)
```

Parameters	windowParent:	The parent application window that is used for centering login dialogues.
	applicationCcowContextModule:	The application's CCOW context module, which the login module should use to read the CCOW context.
	applicationCcowContextBroker:	The application's CCOW context broker, which the login module should use to read the CCOW context.

Field Details

Field VHA_CCOW_LOGON_TOKEN

Description The user context location under which the Kernel token is stored.

See Also: Constant Field Values.

Format `public static final java.lang.String` **`VHA_CCOW_LOGON_TOKEN`**

Field VHA_CCOW_LOGON_NAME

Description The user context location under which the user name is stored.

See Also: Constant Field Values.

Format `public static final java.lang.String` **`VHA_CCOW_LOGON_NAME`**

Field VHA_CCOW_LOGON_VPID

Description The User Context location under which the VPID is stored. The VPID key is not yet supported.

See Also: Constant Field Values.

Format `public static final java.lang.String` **`VHA_CCOW_LOGON_VPID`**

Field VHA_CCOW_USER_CONTEXT_KEYS

Description Array containing the complete set of VHA User Context keys. The VPID key is not yet supported.

Format `public static final java.lang.String[]` **`VHA_CCOW_USER_CONTEXT_KEYS`**

Method Details

Method hasNonNullUserContext

Description This method returns whether or not the context contains at least one user context item.

Format `public static boolean` **`hasNonNullUserContext`**`(java.util.Map contextItems)`

Parameters `contextItems:` Map of context items representing a context.

Output returns: True—If the context has at least one non-null User Context key/value pair.

III. Systems Management Guide

This is the Systems Management Guide section of this supplemental documentation for the SSO/UC-related software. It is intended for use in conjunction with the SSO/UC Project (Iteration 1). It details the technical-related SSO/UC documentation (e.g., implementation and maintenance of the SSO/UC-related software, routines, files, options, interfaces, product security, etc.).

6. Implementation and Maintenance

Information throughout this manual is meant to help IRM in the implementation and maintenance of the SSO/UC-related software.

Namespace

The SSO/UC-related software consists of patches that have been assigned to the following namespaces (listed alphabetically):

- XU—Kernel
- XWB—RPC Broker

NOTE: Kernel (i.e., Kernel Patch XU*8.0*337) is the designated custodial software package for SSO/UC-related software. However, SSO/UC comprises multiple patches and software releases from several VistA/HealtheVet-VistA applications.

REF: For the specific VistA M Server software patches required for the implementation of SSO/UC, please refer to Table 1-2 in Chapter 1, "SSO/UC Project Overview" in this manual.

^XTMP Global

The SSO/UC-related software uses the ^XTMP(<xwb_handle>,0) global to store the Kernel CCOW login token and other temporary variables. The tokens are indexed by the token handle, which is used for retrieval/validation during subsequent SSO login attempts.

Site Parameters

Set the CCOW TOKEN TIMEOUT field (#30.1) in the KERNEL SYSTEM PARAMETERS file (#8989.3) to the length of time (in seconds) that the Kernel CCOW login token is valid. When the current system time is greater than the Kernel CCOW login token create time plus the timeout seconds stored in this field, the Kernel CCOW login token will no longer be valid.

Valid values for this field range from 600 to 28,800 seconds, 0 decimal digits (i.e., 10 minutes minimum to 8 hours maximum). The default value is 5,400 seconds (i.e., 1.5 hours):

- If this value is too small (short), users will be frustrated that the SSO functionality doesn't work optimally.
- If the value is too large (long), there is a chance that the token could be compromised and used to break into the system.

Remote Procedure Calls (RPCs)

The following remote procedure call (RPC) is exported with the SSO/UC-related software:

RPC Name	RPC Description
XUS GET CCOW TOKEN	(Private) This RPC is called in order to sign on, set up, and return the Kernel CCOW login token handle from the VistA M Server.

Table 6-1. SSO/UC-related RPC list

Disabling Single Sign-On/User Context (SSO/UC)

For sites whose policy is *not* to allow the kinds of SSO-based logins supported by the SSO/UC Project (Iteration 1), the User Context-based SSO can be disabled by doing either of the following:

- **Mark the User subject as "unshared" in the Sentillion Vergence Context Vault so that the User subject instance is kept separate for all application instances.** This is how the Sentillion Vergence Context Vaults were initially configured when VHA first procured them for Patient Context (i.e., User Context was specifically disabled).
- **Do *not* grant secure access in the Sentillion Vergence Context Vault to the application passcode used by the login component/classes.** Without the application passcode, the login component/classes *cannot* establish a secure binding to the User Context. This failure triggers a standard, non-SSO login process:
    1. The login component/classes do not find a User Context.
    2. The login component/classes prompt the user for their Access and Verify code credentials.
    3. The application logs in; and no User Context is set.

VistA M Server—Configuring User Accounts

The main user sign-on routine has been modified to accept and process a Kernel CCOW login token. This validates the user for sign-on to CCOW-enabled and SSO/UC-aware applications.

Kernel CCOW Login Token Expiration

Users running VistA CCOW-enabled and SSO/UC-aware applications on a client workstation are automatically authenticated and logged onto a VistA M Server if a valid (i.e., *not* expired) Kernel CCOW login token for that user exists on the VistA M Server. However, if a user launches a VistA application after the Kernel CCOW login token has expired, they will be prompted to enter their Access and Verify codes.

NOTE: Applications that were started/authenticated *prior* to the Kernel CCOW login token expiration period continue to run and are *not* affected once the expiration period is reached.

The Kernel CCOW login token is valid from a minimum of 600 seconds to a maximum of 28,800 seconds (i.e., 10 minutes to 8 hours) from when the user first authenticated via Kernel on the VistA M Server. The default value is 5,400 seconds (i.e., 1.5 hours). This default value is a compromise between wanting to provide as rapid a Kernel CCOW login token expiration as possible for security reasons, versus the need for a SSO session to last long enough in order to be useful to the user.

To change the expiration time, IRM can change the value stored in the CCOW TOKEN TIMEOUT field (#30.1) in the KERNEL SYSTEM PARAMETERS file (#8989.3).

REF: For more information on the CCOW TOKEN TIMEOUT field, please refer to the "Files and Fields" topic that follows in this chapter.

Files and Fields

The following field is exported with the SSO/UC-related software:

File Number	File Name	Field Name	Field Number	Field Description
8989.3	KERNEL SYSTEM PARAMETERS	CCOW TOKEN TIMEOUT	30.1	This field stores the value of the length of time (in seconds) that the Kernel CCOW login token is valid. When the current system time is greater than the Kernel CCOW login token create time plus the timeout seconds stored in this field, the Kernel CCOW login token will no longer be valid. Valid values for this field range from 600 to 28,800 seconds, 0 decimal digits (i.e., 10 minutes minimum to 8 hours maximum). The default value is 5,400 seconds (i.e., 1.5 hours): • If this value is too small (short), users will be frustrated that the SSO functionality doesn't work optimally. • If the value is too large (long), there is a chance that the token could be compromised and used to break into the system.

Figure 6-1. Field exported with the SSO/UC Project (Iteration 1)

Global Mapping/Translation, Journaling, and Protection

There are *no* special global mapping/translation, journaling, and protection instructions for the SSO/UC-related software.

Routines

This topic contains a list of the routines modified and exported with the SSO/UC-related software. A brief description of the routines is provided.

Routine Name	Routine Description
XUS	This is the main VistA M Server user sign-on routine. It was modified to accept Kernel CCOW login tokens.
XUSRB	This is the main sign-on routine for the RPC Broker. It was modified to accept Kernel CCOW login tokens.
XUSRB4	This routine is called by an RPC. It was modified to build a Kernel CCOW login token that is returned to the calling application.
XWBSEC	This routine was modified to add the XUS GET CCOW TOKEN to list of allowed RPCs.

Table 6-2. SSO/UC-related software routine list

Exported Options

There are *no* options exported with the SSO/UC-related software.

Archiving and Purging

There are *no* special archiving or journaling instructions for the SSO/UC-related software.

Callable Routines/Methods

The SSO/UC-related software provides the following callable routine entry points (i.e., Application Program Interfaces [APIs]) and methods that are available for general use. All other SSO/UC-related APIs are used internally by Kernel on the server and the RPC Broker component or VistALink classes on the client.

Software	Entry Point/Method	Brief Description	For More Information
Kernel	PROD^XUPROD([force])	This function is called by applications to check and see if the application is running in a Production or a Test account.	See the "PROD^XUPROD(): Production Vs. Test Account" API description in Chapter 4 in this manual.
RPC Broker	IsUserCleared	This method is used to determine whether or not the User Context has changed to null in RPC Broker-based CCOW-enabled and SSO/UC-aware applications.	See Step #4b in the "RPC Broker-based Client/Server Applications" topic in Chapter 3 in this manual.
VistALink	hasNonNullUserContext	This method returns whether or not the context contains at least one user context item in VistALink-based CCOW-enabled and SSO/UC-aware applications.	See the "Method Details" topic in the "CallbackHandlerSwingCCOW Class" topic in Chapter 4 in this manual.

Table 6-3. Callable routines/methods for the SSO/UC Project (Iteration 1)—Listed by software application

External Relations

VistA Software Requirements

The SSO/UC software relies on the following VistA software to run effectively (listed alphabetically):

Software	Version	Software/Patch Information
Kernel	8.0	Server software—Fully patched.
Kernel Toolkit	7.3	Server software—Fully patched.
MailMan	8.0	Server software—Fully patched.
RPC Broker	1.1	Client/Server software—Fully patched.
VistALink	1.5	Client/Server software—Fully patched.
VA FileMan	22.0	Server software—Fully patched.

Table 6-4. External Relations—VistA **software**

NOTE: Kernel (i.e., Kernel Patch XU*8.0*337) is the designated custodial software package for SSO/UC-related software. However, SSO/UC comprises multiple patches and software releases from several VistA/HealtheVet-VistA applications.

REF: For the specific VistA M Server software patches required for the implementation of SSO/UC, please refer to Table 1-2 in Chapter 1, "SSO/UC Project Overview" in this manual.

COTS Software Requirements

The CCOW-enabled and SSO/UC-aware RPC Broker login component and VistALink login classes compiled into VistA applications interface with the following Commercial-Off-The-Shelf (COTS) software products in order to run effectively (listed alphabetically):

Category	Software	Version	Software/Patch Information
Client	Sentillion Vergence Desktop Components	3.3	Client software—Fully patched. Sentillion software for *programmer-only* client workstations used to develop CCOW-enabled and SSO/UC-aware applications: • Vergence Locator • Context Manager Prox
	Sentillion Software Development Kit (SDK)	Latest	Client software—Fully patched. Sentillion software for *programmer-only* client workstations used to develop CCOW-enabled and SSO/UC-aware applications. Two types of SDK are provided: • COM-based Rich Client SDK—Provided as a COM component. • Java-based Web Client SDK—Intended to CCOW-enable web applications written in Java.
Server	Sentillion Vergence Context Vault	3.3	Server software—Fully patched. Sentillion software for servers called by CCOW-enabled and SSO/UC-aware applications.

Table 6-5. External Relations—COTS software

NOTE: This COTS software and documentation are distributed separately from the VistA/HealtheVet-VistA software and documentation.

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

REF: For more information on the Sentillion COTS software, please refer to Chapter 2, "SSO/UC VistA Applications/Modules," in this manual.

Integration Agreements (IAs)

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.

The SSO/UC-related software is *not* dependent on any agreements.

To obtain the current list of IAs to which Kernel, RPC Broker, or VistALink 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**; or **XWB** or **RPC Broker**; or **XOBS** or **VistALink**.
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 to which Kernel or RPC Broker 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**, **XWB**, or **XOBS**.
7. All current IAs to which the software is a subscriber are listed.

Internal Relations

Relationship of SSO/UC-related Software with VistA

Namespace

The SSO/UC-related software consists of patches that have been assigned to the following namespaces (listed alphabetically):

- XOBS and XOBV—VistALink
- XU—Kernel
- XWB—RPC Broker

NOTE: Kernel (i.e., Kernel Patch XU*8.0*337) is the designated custodial software package for SSO/UC-related software. However, SSO/UC comprises multiple patches and software releases from several VistA/Health*e*Vet-VistA applications.

REF: For the specific VistA M Server software patches required for the implementation of SSO/UC, please refer to Table 1-2 in Chapter 1, "SSO/UC Project Overview" in this manual.

Kernel V. 8.0

In order to develop RPC Broker- or VistALink-based applications so that they are SSO/UC-aware, the following Kernel patches *must* be installed (listed in patch number order):

- Server Patch—XU*8.0*265
- Server Patch—XU*8.0*284
- Server Patch—XU*8.0*337
- Server Patch—XU*8.0*361

RPC Broker V. 1.1

In order to develop RPC Broker-based applications so that they are SSO/UC-aware, the following RPC Broker patches *must* be installed (listed in patch number order):

- Server Patch—XWB*1.1*35
- Client Patch—XWB*1.1*40

VistALink V. 1.5

In order to develop VistALink-based applications so that they are SSO/UC-aware, VistALink V. 1.5 *must* be installed.

Software-wide and Key Variables

The SSO/UC-related software does *not* employ the use of software-wide or key variables.

SACC Exemptions

The SSO/UC-related software does *not* have any Programming Standards and Conventions (SAC) exemptions.

7. Software Product Security

Security Management

There are *no* special legal requirements involved in the use of the SSO/UC-related software.

Mail Groups and Alerts

Mail Groups

The SSO/UC-related software does *not* create or utilize any specific mail groups.

Alerts

The SSO/UC-related software does *not* make use of alerts.

Remote Systems

The RPC Broker login component and VistALink login classes connect and transmit data to local and remote VistA M systems using Transmission Control Protocol/Internet Protocol (TCP/IP), allowing connections between client workstation applications and the VistA M Server. VistA applications can use any remote procedure call (RPC) authorized to the application, if the application is authorized to the signed-on user.

Interfaces

The CCOW-enabled and SSO/UC-aware RPC Broker login component and VistALink login classes compiled into VistA CCOW-enabled and SSO/UC-aware applications interface with the following *non*-VA Commercial-Off-The-Shelf (COTS) products:

- Sentillion Vergence Context Vault
- Sentillion Vergence Desktop Components
- Sentillion Software Development Kit (SDK)

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

REF: For more information on the Sentillion COTS software, please refer Chapter 2, "SSO/UC VistA Applications/Modules," and Table 6-5 in this manual.

Electronic Signatures

There are *no* electronic signatures used within the SSO/UC-related software.

Security Keys

There are *no* specific security keys exported with this version of the SSO/UC-related software.

File Security

There are *no* new file or field security changes associated with the SSO/UC-related software.

REF: For information on the CCOW TOKEN TIMEOUT field (#30.1) in the KERNEL SYSTEM PARAMETERS file (#8989.3), please refer to the "Files and Fields" chapter in this manual.

Official Policies

There are *no* special legal requirements involved in the use of the SSO/UC-related software.

Distribution of the SSO/UC-related software 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 Web address:

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

Glossary

CCOW

"By synchronizing and coordinating applications so that they automatically follow the user's context, the CCOW Standard serves as the basis for ensuring secure and consistent access to patient information from heterogeneous sources. The benefits include applications that are easier to use, increased utilization of electronically available information, and an increase in patient safety. Further, CCOW support for secure context management provides a healthcare standards basis for addressing HIPAA requirements. For example, CCOW enables the deployment of highly secure single sign-on solutions."[2]

CLASSES

An object-oriented term used to describe the building blocks of GUI VistALink-based Java applications. A software object that contains data and code. These classes interact with other classes to create the GUI user application interface.

CLIENT

A single term used interchangeably to refer to the user, the workstation, and the portion of the program that runs on the workstation. This term is typically used in an object-oriented environment, where a client is a member of a group that uses the services of an unrelated group. If the client is on a local area network (LAN), it can share resources with another computer (server).

With respect to the SSO/UC software, client refers to the "requesting server" that is able to connect to a "receiving server," where both servers reside in VistA on the same or on different VistA M systems.

COMPONENT

An object-oriented term used to describe the building blocks of GUI RPC Broker-based Delphi applications. A software object that contains data and code. A component may or may not be visible. These components interact with other components on a form to create the GUI user application interface.

[2] From a white paper titled "Overview of HL7's CCOW Standard"; copyrighted 2001 Health Level Seven, Inc.; author Robert Seliger (Sentillion), Co-Chair, CCOW Technical Committee from the http://www.hl7.org/special/committees/visual/visual.cfm Web site.

DLL

Dynamic **L**ink **L**ibrary. A DLL allows executable routines to be stored separately as files with a DLL extension. These routines are only loaded when a program calls for them. DLLs provide several advantages:

1. DLLs help save on computer memory, since memory is only consumed when a DLL is loaded. They also save disk space. With static libraries, your application absorbs all the library code into your application so the size of your application is greater. Other applications using the same library will also carry this code around. With the DLL, you don't carry the code itself, you have a pointer to the common library. All applications using it will then share one image.

2. DLLs ease maintenance tasks. Because the DLL is a separate file, any modifications made to the DLL will not affect the operation of the calling program or any other DLL.

3. DLLs help avoid redundant routines. They provide generic functions that can be utilized by a variety of programs.

ENCRYPTION

Scrambling data or messages with a cipher or code so that they are unreadable without a secret key. In some cases encryption algorithms are one directional, that is, they only encode and the resulting data cannot be unscrambled (e.g., Access and Verify codes).

HOST

The term Host is used interchangeably with the term Server.

LISTENER

The server process that listens for and accepts incoming connection requests from client applications.

SERVER

With respect to client/server software, server refers to the "receiving server" that sends the results in a message back to the "requesting server," where both servers reside in VistA on the same or on different VistA M systems.

The server is where VistA M-based data and Business Rules reside, making these resources available to the requesting server.

When the requesting server is receiving the results, it is referred to as the "server."

SIGN-ON/SECURITY

The Kernel module that regulates access to the menu system. It performs a number of checks to determine whether access can be permitted at a particular time. A log of signons is maintained.

XML

E**x**tensible **M**arkup **L**anguage. The universal format for structured documents and data on the Web.

REF: For a comprehensive list of commonly used infrastructure- and security-related terms and definitions, please visit the ISS Glossary Web page at the following Web address:

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

For a comprehensive list of acronyms, please visit the ISS Acronyms Web site at the following Web address:

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

Appendix A—SSO/UC Prototype (Pre-release)

The SSO/UC Prototype software served as the model for the final SSO/UC Project (Iteration 1) software. The SSO/UC Prototype demonstration involved the use of the following applications to allow sharing of User (and Patient where applicable) Context on a given VistA system, demonstrating CCOW-based SSO:

- **Kernel—**Kernel is the approved method of authentication for single sign-on in the VHA environment. Kernel was assessed as the most straightforward and timely approach to demonstrating single sign-on in the VA, as this is where users are currently managed. Using Kernel as the authenticator meant that a separate user data store other than the NEW PERSON file (#200) need *not* be created and maintained.
- **Sentillion Vergence Context Vault—**Sentillion's Vergence Context Vault Commercial-Off-The-Shelf (COTS) software was used to store user sign-on credentials in a CCOW User Context, in a similar fashion to how VA is already using the Context Vault to synchronize applications on Patient Context. It was properly configured for User Context, including the required application names and passcodes.

  NOTE: This COTS software is already being installed and configured as part of the Clinical Context Management Project (CCOW) release for Patient Context. However, for SSO/UC, a separate User subject license is required. This license has already been purchased, but it now *must* be installed.

  REF: For more information on the CCOW Package Release, please refer to the Context Management Project (CCOW) release documentation available on the EVS Anonymous Directories.

- **Sentillion Vergence Desktop Components—**Sentillion's Vergence Desktop Components COTS software was used to provide communication/linkage between the CCOW-enabled login components embedded in the CCOW-enabled VistA applications on the client workstation and the Sentillion Vergence Context Vault on the server.

  NOTE: This COTS software is already being installed and configured as part of the Clinical Context Management Project (CCOW) release for Patient Context.

  REF: For more information on the CCOW Package Release, please refer to the Context Management Project (CCOW) release documentation available on the EVS Anonymous Directories.

- **CCOW-enabled Computerized Patient Record System (CPRS) Application—**A test version of this rich client/M Server program (i.e., COM client application developed in Borland Delphi) was enabled for User Context by recompiling the code using the CCOW-enabled RPC Broker component (i.e., TCCOWRPCBroker).
- **CCOW-enabled Care Management (CM) Application—**A test version of this rich client/M Server program (i.e., client application developed in Java) was enabled for User Context by recompiling the code using the CCOW-enabled version of VistALink classes.

- **CCOW Context Monitor—**A test version of this rich client application was created that provided the current CCOW User Context identity and the ability to clear the User Context and logoff all CCOW-enabled and SSO/UC-aware applications.
- **CCOWTiming Application—**This sample rich client/M Server program (i.e., COM client application developed in Borland Delphi) was enabled for User Context by recompiling the code using the CCOW-enabled RPC Broker component (i.e., TCCOWRPCBroker). It tests the amount of time for Remote Procedure Calls to be processed by the VistA M Server.
- **CCOW-enabled Telnet Application—**This sample rich client/M Server Telnet program was written in-house using Borland Delphi and was enabled for User Context by recompiling the code using the CCOW-enabled RPC Broker component (i.e., TCCOWRPCBroker). This application allows users to access legacy Roll-and-Scroll applications (e.g., List Manager) and still provide Single Sign-On/User Context for those applications.

  NOTE: This sample Telnet application will *not* be released as part of the SSO/UC Project (Iteration 1).

- **J2EE Web Application—**An additional model explored as part of the prototype was an Oracle 9iAS J2EE Web-based application.

  NOTE: The Kernel Authentication and Authorization for Java 2 Enterprise Edition (KAAJEE) project is currently underway. It will provide Kernel authentication and authorization capability for Web-based applications (i.e., Java client web-based applications) in a Web-based application server environment (e.g., Oracle 9iAS, WebLogic). In the future, via a separate project, these Web-based applications will also be made CCOW-enabled and SSO/UC-aware.

  REF: For more information on KAAJEE, please refer to the *Kernel Authentication and Authorization for Java 2 Enterprise Edition (KAAJEE) Installation Guide* and *KAAJEE Deployment Guide*.

All prototype applications were able to demonstrate SSO, authenticating against Kernel and enabled through CCOW User Context. The SSO/UC Prototype implementation worked as follows:

1. The user is initially authenticated via Kernel and is given a Kernel CCOW login token for logging on to the system.
2. The Kernel CCOW login token is stored in the Sentillion Vergence Context Vault.
3. Sign-on by the user to other CCOW-enabled and SSO/UC-aware applications is then made through use of the User information in the Sentillion Vergence Context Vault, in the same manner that Patient Context is managed.

The following User Context data was identified to support single sign-on in the SSO/UC Prototype solution:

```
//The VistA Domain
CCOW_LOGON_ID = 'user.id.logon.vistalogon';        //CCOW
//The VistA Token
CCOW_LOGON_TOKEN = 'user.id.logon.vistatoken';     //CCOW
//The VistA user Name
CCOW_LOGON_NAME = 'user.id.logon.vistaname';       //CCOW
//The User Name in VistA
CCOW_NAME_VistA = 'user.id.logon.vpid';
```

Figure A-1. User Context data for SSO/UC Prototype solution

From the prototype development efforts, the following recommendations were made to proceed with the implementation of SSO/UC:

- Use CCOW User Context to provide SSO for rich client/M Server RPC Broker- and VistALink-based applications.
- Use Kernel authentication for CCOW-enabled and SSO/UC-aware rich client applications and server-based Web applications (e.g., Oracle 9iAS-based J2EE applications), as is already in place for existing VistA applications.

  Using Kernel reduces the number of competing user authentication systems and user stores (i.e., NEW PERSON file [#200]). Avoiding an additional user store, simplifies the migration to any future AA solutions.
- Revisit CCOW-based SSO for J2EE application server-based Web applications in the future.

Index