VistALink

VERSION 1.0

Developer / System Manager Manual

October 2, 2003

Department of Veterans Affairs
VistA Health Systems Design & Development (HSD&D)

Revision History

Table 1, below, summarizes the *VistALink Developer / System Manager* Manual revision history.

Date	Revision	Description	Author(s)
April 18, 2003	N/A	Initial draft content	Foundations team
May 23, 2003	.9	Revisions from CPRS-R, VHA Technical Writers and VHA Testing received / implemented	Foundations team / Reviewers
October 2, 2003	1.0	Revisions from VistALink Development Team received / implemented	Foundations team

Table 1: VistALink Developer / System Manager Manual Revision History

Contents

Figures

Tables

Using This Manual

This manual offers advice and instructions regarding the use of VistALink and the functionality it provides for Veterans Health Information Systems and Technology Architecture (VistA) as a whole.

- 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/dialogs) and computer source code are shown in a *non*-proportional font and enclosed within a box. Also included are Graphical User Interface (GUI) Microsoft Windows images (i.e., dialogs or forms).
    - ➢ User responses to online prompts will be boldface type.
    - ➢ The word "**Enter**" in snapshots further prompts the user to press the **Enter** or **Return** key on their keyboard.
    - ➢ Author comments are displayed in italics or as "callout" boxes.

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

Common Terms

The terms and their descriptions in **Table 2**, below, may be helpful while reading this manual.

Term	Description
Adapter / Connector	Exemplified by the Java 2 Platform, Enterprise Edition (J2EE) Connector Architecture (J2EE Connectors for short), this represents a uniform way to integrate J2EE application servers with Enterprise Information Systems (EIS).
Authentication	Verifying the identity of the end-user.
Authorization	Checking the permissions of a user to allow or disallow the performance of some function.
Classpath	Where Java classes must be in order for the JVM to load them. *See JVM definition below.*
Client	A single term used interchangeably to refer the *client* workstation (i.e., PC), and the *client* portion of the program that runs on the workstation.
J2EE	Java 2 Platform, Enterprise Edition. For more relevant detail, see the above definition for *Adapter / Connector.*
J2SE	Java 2 Standard Edition. The blueprint for building Java applications.
JAAS	Java Authentication and Authorization Service. This Java package enables services to authenticate and enforce access controls upon users.
JAR	Java Archive. A file format based on the popular ZIP file format, used for aggregating many files into one.

JAVA	An object-oriented language similar to C++, but simplified to eliminate language features that cause common programming errors.
JRE	Java Runtime Environment, also known as Java Runtime, is part of the Java Development Kit (JDK), a set of programming tools for developing Java applications.
JVM	Java Virtual Machine. It interprets compiled Java binary code (called bytecode) for a computer's processor (or "hardware platform") so that the computer can perform a Java program's instructions.
M Server	The computer where MUMPS or M data and Remote Procedure Calls (RPCs) reside.

Table 2: Common VistALink Terms

How to Obtain Technical Information Online

ℹ Methods of obtaining specific technical information online will be indicated in this manual (where applicable) under the appropriate topic.

Assumptions About the Reader

This manual is written with the assumption that readers have experience with the following:

- VistA computing environment (e.g., Kernel Installation and Distribution System [KIDS])
- VA FileMan data structures and terminology
- Microsoft Windows
- Java development environment
- M programming language
- Java programming language

This manual makes no attempt to explain how the overall 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) for a general orientation to VistA. For example, go to the System Design & Development (SD&D) Home Page at the following web address:

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

Reference Materials

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

- *VistALink Installation Guide,* available at http://www.va.gov/vdl/.
- *VistALink Technical Manual and Package Security Guide,* also available at http://www.va.gov/vdl/.
- http://vista.med.va.gov/migration/foundations/Foundindex.htm

The Foundations page (available at the URL above) provides announcements, additional information (e.g., Frequently Asked Questions [FAQs] or advisories), documentation links, archives of older documentation and software downloads.

VistALink documentation is made available online, on paper and in Adobe Acrobat Portable Document Format (.PDF). A .PDF must be read using the Adobe Acrobat Reader (i.e., ACROREAD.EXE), which is freely distributed by Adobe Systems Incorporated at the following URL or Web address:

http://www.adobe.com/

For more information on the use of the Adobe Acrobat Reader, please refer to the "Adobe Acrobat Quick Guide," also available at the Adobe URL above.:

DISCLAIMER: The appearance of external hyperlink references in this manual does not constitute endorsement by the Department of Veterans Health Administration (VHA) of this Web site or the information, products or services contained therein. The VHA 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 VHA Intranet Service.

Introduction

This guide defines one of the three (3) VistALink v1.0 documentation deliverables:

1. Installation Guide
2. Developer / System Manager Manual
3. Technical Manual and Package Security Guide

*The **VistALink Installation Guide** details all the steps required for setting up the VistALink M software components. The **VistALink Developer / System Manager Manual** gives nationwide VHA application modernization teams background information and specific instructions focused on VistALink as a tool. For questions concerning the architecture and construction of VistALink, the **VistALink Technical Manual and Package Security Guide** provides qualified answers, as well as foundational information. Together, these three publications document the current state of VistALink and anticipate upcoming development.*

Background Information. VistALink is a transport layer that allows Java to communicate with M remote procedures. VistALink is completely based on standard technologies, both on the Java and M side.

Architectural Scope. VistALink v1.0 is designed to work with a standalone J2SE application. VistALink will be used by other VistA rehosting projects as a transport layer between Java and M. VistALink implements the J2EE Connector Architecture 1.0 standard – v1.0 of VistALink implements only the non-managed (i.e., outside of a J2EE container) version of the J2EE Connectors.

Functionality Scope. VistALink v1.0 provides:

a. Communication capabilities for an M application request from a client J2SE Java application

b. An authenticated communication transport layer from Java to M

c. End-user authentication within Java applications based on M user accounts (J2SE only)

d. Calling RPCs in a secure environment

VistALink FAQs.

For general or frequently asked questions (FAQs) about VistALink, please refer to the following web site:

http://vista.med.va.gov/migration/foundations/FAQ.htm

Part I – Development Workstation and Sample Application Installation

Chapter 1: Installing VistALink on a Developer Workstation

1.1. Overview

This version of VistALink is designed for the J2SE development environment and intended for Java client to M server development. VistALink libraries can be used from within any Java-compatible, Integrated Development Environment (IDE). Examples of Java-compatible IDEs include Eclipse and JDeveloper.

1.2. Installation

1.2.1. Distribution Structure

The VistALink distribution Zip file holds the directories and files detailed below.

<root>	Contains the Readme.txt and ReleaseNotes
allSrc	All VistALink source code
doc	VistALink v1.0 manuals
jars	VistALink Java Archive (JAR) library files
javadoc	API documentation in “javadoc” format
m	KIDS distribution for the M server routines
samples	Source code for the sample application

1.2.2. Installation Instructions

To install VistALink on a developer workstation for development use:

1. Unzip the VistALink distribution Zip file on your workstation.
2. Add the JAR files below to the Java classpath for each development project with which you are using VistALink.
    - vljConnector_1.0.jar
    - vljFoundationsLib_1.0.jar
    - vljSecurity_1.0.jar

    JAR file version numbers may vary depending on the VistALink distribution version number.
3. Obtain, load on your system and add the supporting libraries required by VistALink, as detailed in the next subsection.

1.2.3. Supporting Libraries Required by VistALink

Developing applications with VistALink requires that some supporting libraries be in your classpath. These libraries are:

- **jaxen-full.jar** (source: JAXEN project, version 1.0-FCS, http://prdownloads.sourceforge.net/jaxen/jaxen-1.0-FCS.zip?download
- **saxpath.jar** (source: JAXEN project, version 1.0-FCS)
- **j2ee.jar** (source: Sun Microsystems, version 1.3.1, http://java.sun.com/j2ee/sdk_1.3/)
- **JAXP-compliant XML parser and XSLT transformer, such as xerces.jar and xalan.jar** (please see the JAXP specification for more details)
- **log4j.jar** (source: log4j apache logging framework, available at http://jakarta.apache.org/log4j

Note: To obtain the j2ee.jar library file, it is necessary to install the complete J2EE version 1.3 SDK on your machine. Once you do that, get the j2ee.jar file from the "lib" subdirectory of the directory you installed the J2EE 1.3 SDK in. Save this file to the location where you are keeping the other supporting libraries (for example), and then you can de-install the J2EE 1.3 SDK.

1.2.4. Java Classpath Considerations

When setting up any given application, the libraries (jar files), classes, configuration files and other resources used by an application need to be on the application's java *classpath*. There are a number of ways to set the java classpath for an application, including:

- IDE project-specific classpath
- CLASSPATH environment variable (Windows), workstation-wide scope
- CLASSPATH environment variable (Windows), set within a batch file, scope of the batch file/command window execution only
- -classpath (***or*** *–cp*) command line argument to the Java Virtual Machine

For more information about Java's *classpath* and how to set it, please see:

- *Setting the class path*, Sun Microsystems, Inc., http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/classpath.html

1.2.5. Additional Installation References

For complete instructions on installing the VistALink package for the M server side, see the *VistALink Installation Guide.* Chapter 2 of the *Installation Guide* includes the **VistALink IRM Preparation and Installation Checklist**, created to assist IRM programmers with the installation of VistALink v1.0 and the testing of the sample Java applications.

1.3. Public VistALink APIs Documented in Javadoc

Included in the VistALink distribution is standard "Javadoc" API documentation for the various Java classes that make up the public VistALink programming API. The VistALink classes documented in the VistALink Javadoc comprise VistALink's public APIs. These APIs may be used under the conditions listed in the Javadoc documentation.

Other VistALink classes not documented in the VistALink javadoc are not part of the supported VistALink API.

To access this documentation:

1. Go to the location to which you unzipped the VistALink distribution on your workstation.
2. Go to the "javadoc" subdirectory.
3. Load "index.html" in your web browser.

For more information about the JavaDoc documentation format, please see http://java.sun.com/j2se/javadoc/.

Chapter 2: Sample Applications

2.1 Overview

Three sample applications are supplied in vljSamples_1.0.jar, with the following class names:

- VistaLinkRpcSwingTester (full featured Swing-based listener testing application)
- VistaLinkRpcSwingSimple (simplified Swing-based development example)
- VistaLinkRpcConsole (a console-based application)

2.2 About Installing J2SE Applications

Running any Java 2 Standard Edition (J2SE) application consists first of:

- Installing the correct Java Runtime Environment (JRE)
- Having all supporting Java libraries available
- Setting up the actual application
- Setting up the application classpath
- Setting up any configuration files required by the application

Assumptions: These instructions are written from the point of view of setting up the sample application on a Windows workstation. However, there is nothing about VistALink particularly tied to the Windows client environment, as VistALink is a 100-percent pure Java application.

2.3 Set Up and Run the Sample Applications

2.3.1 JRE Setup

VistALink requires the 1.4.1 or higher J2SE Java Runtime Environment (JRE) or Java Development Kit (JDK) to be installed on the client workstation.

2.3.2 Java Library Installation

VistALink requires certain supporting libraries to be available on the client workstation.

1) Download and install the 1.3.1 J2EE SDK (http://java.sun.com/j2ee/sdk_1.3/)

2) Create a directory to hold the Java libraries required for VistALink, (e.g.) **c:\javalib**.

3) Copy the following files to that directory:

- **j2ee.jar** (source: directory you installed the J2EE 1.3.1 runtime in)
- **jaxen-full.jar** (source: JAXEN, version 1.0-FCS, http://prdownloads.sourceforge.net/jaxen/jaxen-1.0-FCS.zip?download)
- **xerces.jar** (source: xerces) Or other JAXP-compliant XML parser.

- **saxpath.jar** (source: JAXEN)
- **log4j-1.2.7.jar** (source: Log4J, http://jakarta.apache.org/log4j/docs/download.html -- higher versions are OK, e.g., 1.2.8, which is what they have available now)

4) You can uninstall the 1.3.1 J2EE SDK now. You just needed the j2ee.jar file.

JDK 1.4 includes a default JAXP-compliant XML parser and an XSLT transformer. So, VistALink should require no additional JAXP configuration to function on JDK 1.4. In case a need arises to configure a JAXP XML parser or an XSLT transformer, follow the steps detailed in subsection 2.3.8 at the end of this chapter.

2.3.3 VistALink Library Installation

Copy the following files from the VistALink distribution to the same directory on the workstation, for example, c:\javalib.

- vljConnector_1.0.jar
- vljFoundationsLib_1.0.jar
- vljSecurity_1.0.jar

2.3.4 Grant Yourself Kernel Access to the Sample Application

The Kernel "B"-type option “XOBV VISTALINK TESTER” was created as part of the M-side KIDS install. To run the sample application, you will need to grant yourself access to the "XOBV VISTALINK TESTER" on the M server to which you will be connecting (unless you already have Kernel programmer access on the M server).

Note: For more information on granting yourself access to RPCs, please see the RPC Broker Systems Manual at http://www.va.gov/vdl/**.**

2.3.5 Install the Sample Application Files

1) Create a directory, e.g., c:\Program Files\vistalink\samples, for the sample application.

2) Copy the contents of the \samples folder in the distribution file to c:\Program Files\vistalink\samples.

2.3.6 Set Classpath and Java Locations and Run the Sample Applications

Three batch files are supplied in the samples folder of the distribution, one to run each of the three sample applications:

- runRpcConsole.bat (runs VistaLinkRpcConsole)
- runSwingSimple.bat (runs VistaLinkRpcSwingSimple)
- runSwingTester.bat (runs VistaLinkRpcSwingTester)

In addition, a fourth batch file (setVistaLinkEnvironment.bat) is supplied that sets the classpath and the location of the Java.exe executable to use on your workstation. This fourth batch file is called by each of the three batch files noted in section 2.3.6 on the previous page. So to configure the classpath and java executable location for your workstation, you need modify only this one file. The content of this file, as distributed, is:

```
REM -- you will need to adjust the locations of the various jars and
REM    other files to match the locations of these files on your
REM    system.
REM
REM -- set the directory location containing java.exe executable
REM -- (don't include the \bin subdirectory)
set JAVA_HOME=c:\j2sdk1.4.1_02
REM -- if using a JRE, above might be more like (depending on version):
REM -- set JAVA_HOME=c:\program files\java\j2re1.4.1_02
REM
REM clear CLASSPATH and set CLASSPATH for J2EE
set CLASSPATH=c:\javalib\j2ee.jar
REM
REM -- path for JAXEN libraries
set CLASSPATH=%CLASSPATH%;c:\javalib\jaxen-full.jar
set CLASSPATH=%CLASSPATH%;c:\javalib\saxpath.jar
set CLASSPATH=%CLASSPATH%;c:\javalib\xerces.jar
REM
REM -- path for Log4J
set CLASSPATH=%CLASSPATH%;c:\javalib\log4j-1.2.8.jar
REM
REM -- paths for VistaLink (replace 1.0.0.101 with version #)
set CLASSPATH=%CLASSPATH%;c:\javalib\vljConnector_1.0.jar
set CLASSPATH=%CLASSPATH%;c:\javalib\vljFoundationsLib_1.0.jar
set CLASSPATH=%CLASSPATH%;c:\javalib\vljSecurity_1.0.jar
REM
REM -- path for VistaLink sample app  (replace 1.0.0.101 with
version
REM #) -- (assumes the samples jar is in the current directory)
set CLASSPATH=%CLASSPATH%;./vljSamples_1.0.jar
```

So to run the sample applications:

1) Modify the jaas.config file supplied with the distribution's samples folder to have the settings needed to connect to your M system.

Note: runRpcConsole.bat and runSwingSimple.bat are hard-coded to load a configuration named DemoServer from the jaas.config file. Either modify the DemoServer configuration with the settings needed for your M system, or, if you use a different configuration name, modify runRpcConsole.bat and runSwingSimple.bat to use your configuration name (the -s parameter at the end of the command line that launches the application.)

2) Modify the setVistaLinkEnvironment batch file to match the location of the java executable to use on your workstation. You may have multiple Java Runtime Environments (JREs) or Java Development Kits (JDKs) installed on your workstation. Choose which one to use (it should be version 1.4.1 or higher).

In the setVistaLinkEnvironment.bat file, replace the setting for the JAVA_HOME environment variable with the location to use on your system, e.g.:

```
REM -- set the directory location containing java.exe executable
REM -- (don't include the \bin subdirectory)
set JAVA_HOME=c:\j2sdk1.4.1_02
```

If you wish to verify that you have correctly modified the batch file, look in the bin directory of the JAVA_HOME environment variable and use the `java –version` command to determine what version of the JRE you are running. Use the example below as a guide.

```
C:\>CD\j2sdk1.4.1_02\jre\

C:\j2sdk1.4.1_02\jre>CD BIN

C:\j2sdk1.4.1_02\jre\bin>java -version

java version "1.4.2"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM (build 1.4.2-b28, mixed mode)
```

3) Modify the setVistaLinkEnvironment batch file to match the locations of the various supporting library jar files needed to run the sample application. You need to specify the locations of each of the J2EE, JAXEN, Log4J, and VistaLink library jar files. Each entry added to the CLASSPATH variable needs to be modified to match the file name and location of the corresponding library on your system, as you installed them above. For example:

```
REM clear CLASSPATH and set CLASSPATH for J2EE
set CLASSPATH=./j2ee.jar
```

4) runSwingTester.bat (the main sample application, designed to demonstrate VistALink functionality and test server connectivity) can be run directly now. Launch the batch file by double-clicking on it or run it in a command window. Check for errors in the command-window output in case the application is unable to launch; if this happens, the most likely culprit is that one of the locations set in the batch file is incorrect.

5) runSwingSimple.bat (a simpler Swing application that is a better "programming example" program because it lacks the "bells and whistles" of SwingTester) passes a command line parameter to specify which configuration in the jaas.config file should be used to connect to. You should modify the default setting in the batch file if you are not connecting to the default configuration (DemoServer).

6) runRpcConsole.bat is a console-only sample application. In addition to requiring a command-line parameter to specify the JAAS configuration to connect to, is also dependent on passing an access and verify code on the command line (unless the defaults embedded in the application work; they probably will not). You can pass access and verify code in with additional -a and -v command-line parameters.

2.3.7 Optional: To Enable Log4J Logging:

1. Assume c:\Program Files\vistalink\samples is a current directory.

2. Folder c:\Program Files\vistalink\samples\props contains a sample log4jconfig.xml configuration file with various log4j configuration options.

3. Each sample application will try to load log4j configuration from the file named "props\log4jconfig.xml" relative to current directory. Therefore c:\Program Files\vistalink\samples\props\log4jconfig.xml will be loaded.

4. The log4jconfig.xml file within the c:\Program Files\vistalink\samples\props\ folder contains extensive information on various log4j configuration options. Look at this simple example of a log4jconfig.xml file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

  <appender name="myConsoleAppender1" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%-4r [%t] %-5p class %C method %M
line number %L category %c %x - %m%n"/>
    </layout>
  </appender>

  <root>
    <priority value ="info" />
    <appender-ref ref="myConsoleAppender1"/>
  </root>

</log4j:configuration>
```

5. When you run the sample application, you should see "logger" output for debug and error information being displayed on the console window (the window in which you are starting up the application).

Note: an example log4J properties file is provided in the "samples" folder in the distribution zip file.

2.3.8 Optional: JVM Command-Line Parameters – JAXP XML Parser Implementations

VistALink allows you to use any JAXP-compatible XML parser. To enable this support, when you launch an application that uses VistALink, you need to pass some command line parameters to the Java Virtual Machine (JVM) setting some system properties. Those system properties are:

- javax.xml.parsers.SAXParserFactory
- javax.xml.parsers.DocumentBuilderFactory
- javax.xml.transform.TransformerFactory

For example, if you use Xerces as your XML parser and Xalan as your XSLT processor, you would pass the following system properties to the JVM using the -D command line options:

```
java myapp -Djavax.xml.parsers.SAXParserFactory=org.apache.xerces.jaxp.SAXParserFactoryImpl \
-Djavax.xml.parsers.DocumentBuilderFactory=org.apache.xerces.jaxp.DocumentBuilderFactoryImpl \
-Djavax.xml.transform.TransformerFactory=org.apache.xalan.xsltc.trax.SmartTransformerFactoryImpl
```

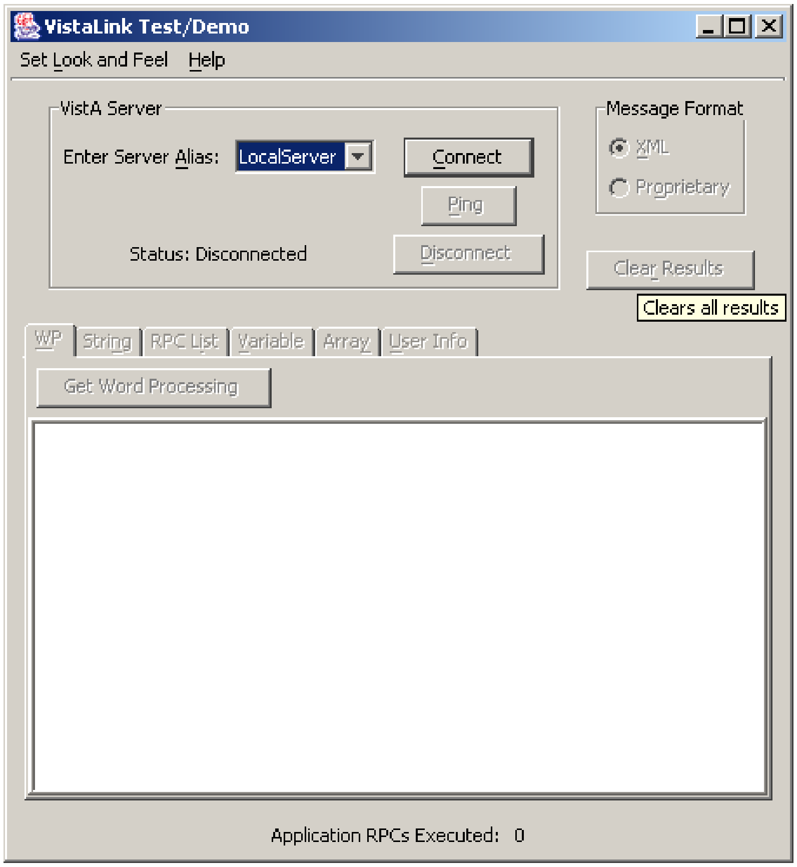
By default, the sample application will use the Xerces as the XML parser and Xalan as the XSLT processor.

Chapter 3: Testing the Listener

3.1 Using the Swing Sample Application

This version of VistALink includes a diagnostic tool for the client workstation (**Figure 1**, below).

Note: The instructions for installing this tool appear in the previous chapter.



Enter Server Alias: LocalServer

Status: Disconnected

● XML

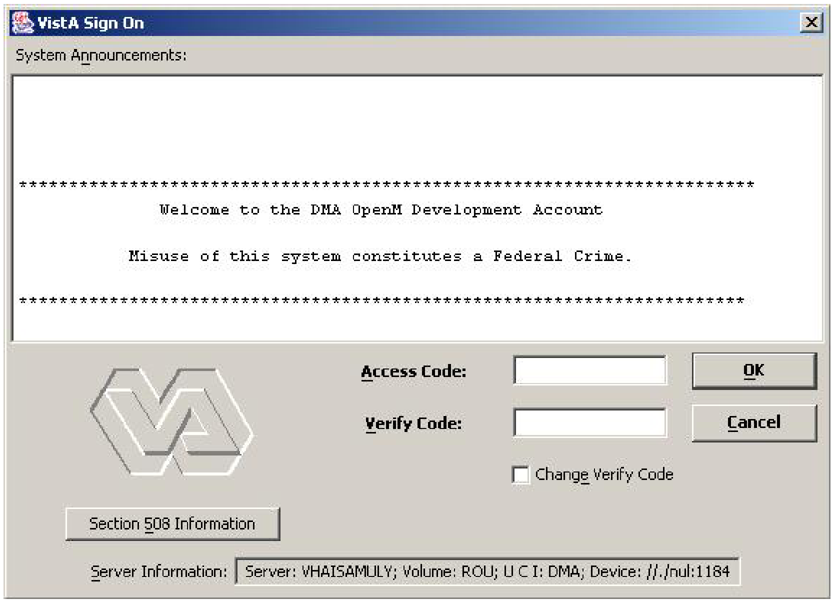
○ Proprietary

Application RPCs Executed: 0

Figure 1: RPC VistALink Connection Diagnostic Program

This tool can be used to verify and test the VistALink client/server connection and signon process. Use the instructions beginning on the next page to work with this tool.

1. Click **Connect** on the **Access / Verify Code** interface shown in **Figure 1**. The interface shown in **Figure 2**, below, will display.

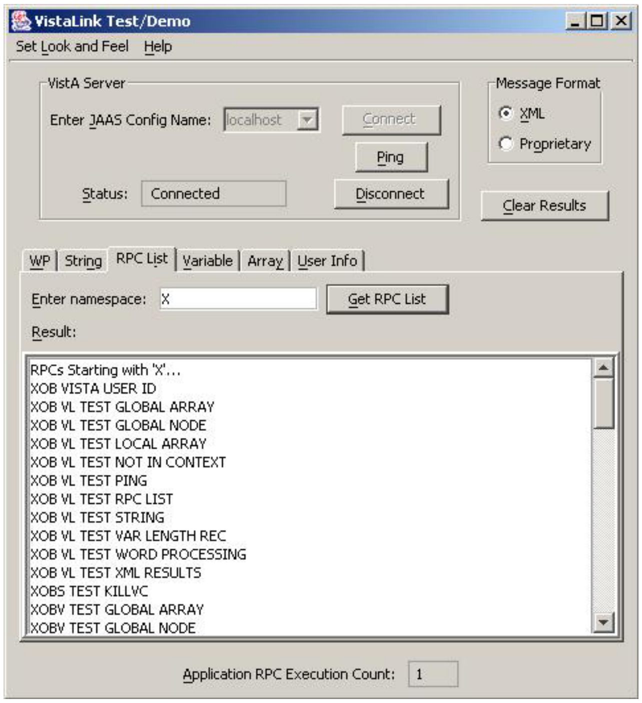


```
******************************************************************************
              Welcome to the DMA OpenM Development Account

           Misuse of this system constitutes a Federal Crime.

*****************************************************************************
```

Server Information: Server: VHAISAMULY; Volume: ROU; U C I: DMA; Device: //./nul:1184

Figure 2: Access / Verify Code Entry

2. Enter the Access / Verify code pair you have been assigned. Click **OK**. An interface with multiple tabs will display. Click on the **RPC List** tab. Type "**X**" in the **Enter namespace** box. Then click **Get RPC List** to display the information in **Figure 3**, below.



Enter JAAS Config Name: localhost

Status: Connected

Enter namespace: X

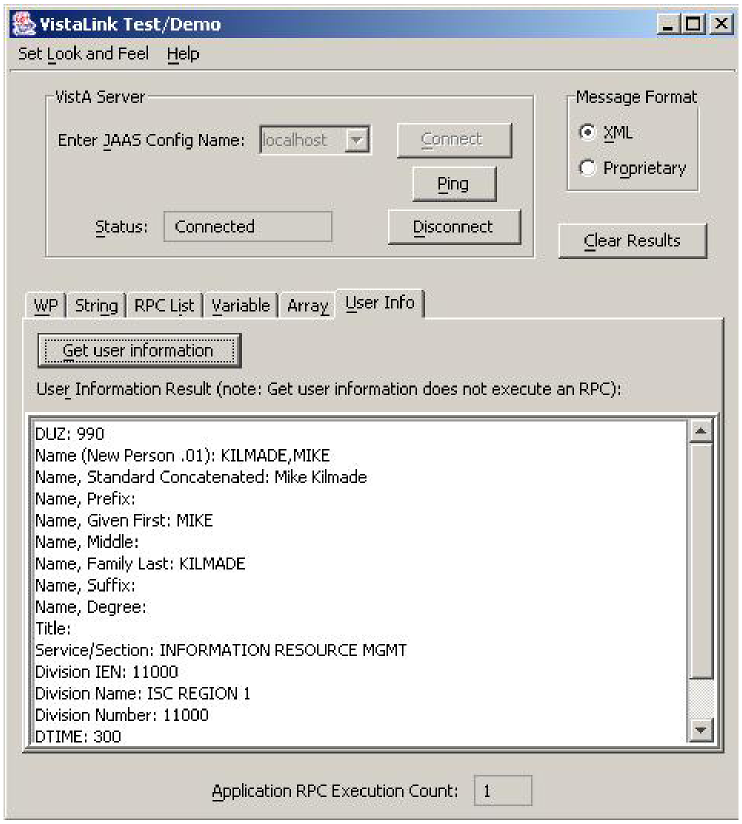
```
RPCs Starting with 'X'...
XOB VISTA USER ID
XOB VL TEST GLOBAL ARRAY
XOB VL TEST GLOBAL NODE
XOB VL TEST LOCAL ARRAY
XOB VL TEST NOT IN CONTEXT
XOB VL TEST PING
XOB VL TEST RPC LIST
XOB VL TEST STRING
XOB VL TEST VAR LENGTH REC
XOB VL TEST WORD PROCESSING
XOB VL TEST XML RESULTS
XOBS TEST KILLVC
XOBV TEST GLOBAL ARRAY
XOBV TEST GLOBAL NODE
```

Application RPC Execution Count: 1

Figure 3: Get RPC List

You can also use the Diagnostic Program to obtain user information.

1. Click on the **User Info** tab in the interface shown in **Figure 4**, below.



Enter JAAS Config Name: localhost

Status: Connected

● XML

○ Proprietary

User Information Result (note: Get user information does not execute an RPC):

```
DUZ: 990
Name (New Person .01): KILMADE,MIKE
Name, Standard Concatenated: Mike Kilmade
Name, Prefix:
Name, Given First: MIKE
Name, Middle:
Name, Family Last: KILMADE
Name, Suffix:
Name, Degree:
Title:
Service/Section: INFORMATION RESOURCE MGMT
Division IEN: 11000
Division Name: ISC REGION 1
Division Number: 11000
DTIME: 300
```

Application RPC Execution Count: 1

Figure 4: User Information

2. Click **Get user information** to display your User data.

3.2 Verifying and Testing the Listener Network Connection

To detect and avoid network problems, do the following:

1. First, make sure you can reach the M server you are trying to connect to through TCP.

   At the DOS/Command prompt type PING nnn.nnn.nnn.nnn to the M server to which you are trying to connect (where nnn.nnn.nnn.nnn equals the IP address of the server). For example:

   ```
   C:\> PING 127.0.0.1 <RET>
   ```

"PINGing" is a way to test connectivity. PINGing sends an Internet Control Message Protocol (ICMP) packet to the server in question and requests a response. It verifies that the server is running and the network is properly configured.

Note: If the M server is unreachable, there is a network problem and you should consult with your network administrator.

2. Verify that the listener *port* the workstation is connecting to is in fact a VistALink listener (and not, for example, a port on which an RPC Broker listener is running). .

3. Telnet from your workstation to the IP address and port of the VistALink listener. On most workstations you can do this simply by entering "telnet ipaddress port" in a command window, e.g.:

   ```
   c:\> telnet 10.21.1.85 8000 <RET>
   ```

   When you connect, press <RET>. If a VistALink listener is running on that port, you should see echoed something similar to:

   ```
   <?xml version="1.0" encoding="utf-8" ?><VistaLink messageType="gov
   .va.med.foundations.vistalink.system.fault" version="1.0" xmlns:xs
   i="http://www.w3.org/2001/XMLSchema-instance"xsi:noNamespaceSchema
   Location="vlFault.xsd"><Fault><FaultCode>Server</FaultCode><FaultS
   tring>System Error</FaultString><FaultActor></FaultActor><Detail><
   Error type="system" code="181001" ><Message><![CDATA[A system erro
   r occurred in M:<SUBSCRIPT>SETMSG+5^XOBVRH]]></Message></Error></D
   etail></Fault></VistaLink>
   ```

   Although there is an error message echoed in this display, the error is due to the fact that we are connecting from telnet rather than from a VistALink client. If an XML message similar to the above is echoed back, then the network connection between your workstation and the VistALink listener at the requested IP address and port is valid, and the problem may lie in the client application configuration, the Java Runtime Environment (JRE), installation, etc.

   If you cannot make the telnet connection, there may be a problem somewhere in the network / firewall / machine TCP configuration.

3.3 Troubleshooting

If you continue to experience problems in testing the listener, examine the areas detailed in the subsections below.

3.3.1 JRE Version

While VistALink requires a Java Runtime Environment (JRE) version of 1.4.1 or greater (when used on a client workstation), the application using VistALink will have its own specific JRE requirements as well; either an equal or higher version than what is needed by VistALink. The driving factor for the JRE that needs to be on the client workstation is the application itself, not the VistALink libraries.

There may be multiple JREs installed on a particular workstation. In this case, you may need to determine which one is being used when you launch the application you are troubleshooting. You may need to look at the shortcut used to launch the application to see if a specific JRE path is specified in an explicit "java" command. You may also need to see if a default JRE is launched when the java command is executed on the workstation (try executing "java -version" in a directory location that does not itself contain the JRE.)

3.3.2 Determining If VistALink Libraries Are Installed on the Client Workstation

If you are examining an application's installation on a client workstation, and want to know if the VistALink libraries are present, look in whatever location a given application places libraries, for the following files:

- vljConnector_x.x.jar
- vljFoundations_x.x.jar
- vljSecurity_x.x.jar

Note: "x.x" is a version number, e.g., 1.0

Different applications may have different versions of libraries (such as VistALink) installed for their own use. This is not necessarily a problem on the client workstation; unlike DLLs, there is no requirement to register java libraries in a system registry. Instead, each application sets its own classpath, and only libraries within an application's specified classpath are loaded by the application. This allows applications with different library version dependencies to co-exist on the same workstation without resulting in a “DLL Hell” situation.

Note: “DLL Hell” is a phrase commonly used in a Windows operations environment.

Part II – Site Management

Chapter 4: Managing the Listener

4.1 Overview

The VistALink TCP listener runs on the M server (although not necessarily in M). Its purpose is to listen on a particular TCP port, accept incoming client connections and spawn off handler jobs to service those connection requests.

4.1.1 VistALink Listeners and Ports

VistALink offers the following listener / Port / IP address possibilities:

- A *single* VistALink listener, running on any available port.
- *Multiple* VistALink listeners running on the same IP address/CPU, but listening on *different* ports.

To run one listener in a Production account and another in a Test account on the same IP address/CPU, you must configure them to listen on different ports (e.g., 8000 for Production and 8001 for Test). If, on the other hand, you are running the listeners on different IP addresses/CPUs, the ports can be the same (e.g., one VistALink listener on every system, listening on port 8000).

While Port 8000 is suggested here, any available port number may be assigned. The listener port, however, must match the port and IP address you specify in the JAAS configurations you set up for your clients.

4.1.2 Differences Between Cache and DSM / VMS

Starting and stopping listeners is different on Cache systems than on DSM/VMS systems.

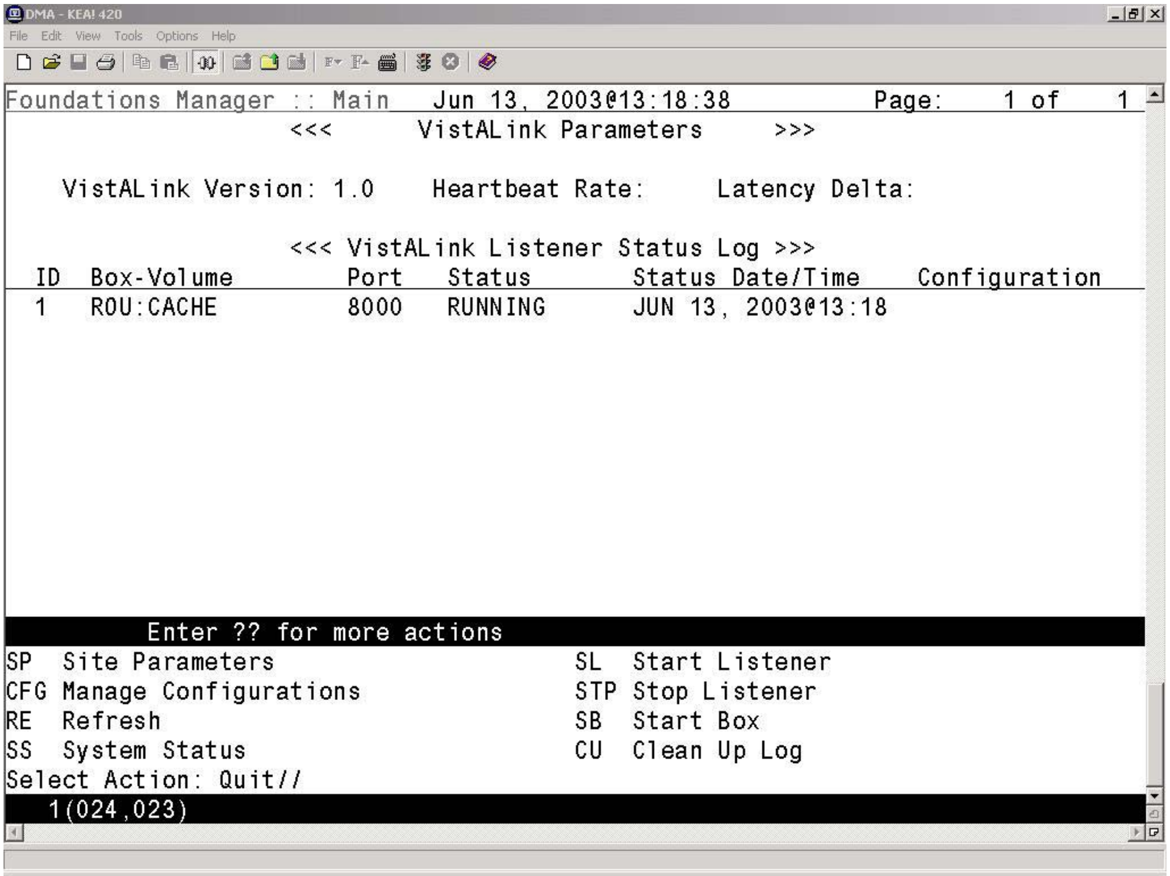
Cache: For Cache systems, listener processes are configured, started and stopped entirely within the M environment. The Foundations Management menu provides several ListMan actions to do this; these particular actions are enabled on Cache systems only.

DSM for OpenVMS: Multi-threaded listeners are externally implemented through the TCP/IP Service for OpenVMS (previously known as UCX). The TCP/IP Service permits multiple TCP/IP clients to connect and run as concurrent processes, up to the limits established by the system. TCP/IP listens on a particular port, and launches a specified VistALink handler process for each client connection. Configuring, starting and stopping listeners is managed entirely through the VMS TCP/IP Service.

4.2 Listener Management for Cache NT and Cache VMS Systems

Access the VistALink system managers' option (XOBU SITE SETUP MENU) using the Operations Management menu, which is available via the Foundations Manager interface (**Figure 5**, below). This option is a ListMan application; for Cache systems, it includes the following executable actions:

SP `Site Parameters`
CFG `Manage Configurations`
RE `Refresh`
SS `System Status`
SL `Start Listener`
STP `Stop Listener`
SB `Start Box`
CU `Clean Up Log`



```
Foundations Manager :: Main   Jun 13, 2003@13:18:38          Page:    1 of    1
                    <<<     VistALink Parameters     >>>

   VistALink Version: 1.0   Heartbeat Rate:    Latency Delta:

                    <<< VistALink Listener Status Log >>>
  ID  Box-Volume        Port   Status        Status Date/Time    Configuration
  1   ROU:CACHE         8000   RUNNING       JUN 13, 2003@13:18









         Enter ?? for more actions
SP  Site Parameters                         SL  Start Listener
CFG Manage Configurations                   STP Stop Listener
RE  Refresh                                 SB  Start Box
SS  System Status                           CU  Clean Up Log
Select Action: Quit//
  1(024,023)
```

Figure 5: Foundations Manager Interface – Cache Systems

Note: The CFG, SL, STP, SB and CU executable actions are not available on DSM systems.

4.2.1 Creating / Editing Listener Configurations

To create or edit listener configuration entries, use the “Manage Configurations” action. The Manage Configurations action first prompts you to select a configuration entry. Then, within the entry, you can configure one or more listeners, as indicated below:

```
Select VistALink LISTENER CONFIGURATION NAME: DEFAULT

NAME: DEFAULT// <Enter>
Select PORT: 8000// <Enter>
  PORT: 8000// <Enter>
  STARTUP: YES// <Enter>
```

Table 3, below, defines the site parameter fields for a given listener entry.

Field	Meaning
Name	This field contains the name of the default VistALink configuration used with the associated BOX-VOLUME PAIR specified in the FOUNDATIONS SITE PARAMETERS file (#18.01).
Port	The port the listener will listen on.
Startup	If the listener should be started when this configuration is started. Set this field to YES. Otherwise, set to NO. (If you want to keep the port in the configuration but temporarily want to not start a listener, you would set field to NO.)

Table 3: Listener Configuration Entries Description Table

4.2.2 To Start All Configured Listeners

To start all listeners (i.e., those configured in the FOUNDATIONS SITE PARAMETERS file to automatically start), use the "Start Box" action shown in **Figure 5** on page 16. This action will start all listeners for the configuration assigned to the current BOX-VOLUME pair.

Note: To configure listeners for automatic startup, please refer to subsection 4.2.1, “Creating / Editing Listener Configurations.”

4.2.3 To Start a Single Unconfigured Listener

To start a listener, use the "Start Listener" (**SL**) action on the Foundations Manager interface and enter the desired port. The action will first check to see if a listener is already listening on the port. If this is true then you will be informed. No damage to the system will occur.

4.2.4 To Stop a Configured or Unconfigured Listener

To stop a running listener, use the "Stop Listener" (**STP**) action and select the desired listener.

Note: The “Stop Listener” action may take up to 60 seconds to actually stop the listener.

4.2.5 How to Schedule Listener Startup at System Startup

The XOBV LISTENER STARTUP option, which starts all VistALink listener configuration operations at one time for the BOX-VOLUME, can be tasked to automatically start all required listener processes upon TaskMan start-up. Examples of when this type of start-up would be required might be after rebooting the system or restarting the configuration.

As part of the VistALink installation for Cache systems, this option should already be scheduled to run at startup.

To automatically start the listeners(s) when TaskMan is restarted, enter the XOBV LISTENER STARTUP option in the OPTION SCHEDULING file (#19.2). Schedule this option with SPECIAL QUEUING set to STARTUP.

You can enter and schedule the required options by using “TaskMan Schedule/Unschedule Options,” as shown in **Figure 6**, below.

```
Select Systems Manager Menu Option: TASKMAN Management

Select Taskman Management Option: SCHedule/Unschedule Options

Select OPTION to schedule or reschedule: XOBV LISTENER STARTUP <Enter>  Start
VistaLink Listener Configuration
         ...OK? Yes// <Enter>  (Yes)


                        Edit Option Schedule
    Option Name:   XOBV LISTENER STARTUP
    Menu Text:    Start VistaLink Listener Configu      TASK ID:
______________________________________________________________________________
___

  QUEUED TO RUN AT WHAT TIME:

DEVICE FOR QUEUED JOB OUTPUT:

 QUEUED TO RUN ON VOLUME SET:

      RESCHEDULING FREQUENCY:

             TASK PARAMETERS:

            SPECIAL QUEUEING:  STARTUP

______________________________________________________________________________
___
```

Figure 6: Automatically Starting Listener(s) Upon TaskMan Restart

4.2.6 Working With the Foundation Site Parameters File

Note: This file is required for both Cache and DSM.

The FOUNDATIONS SITE PARAMETERS file (#18.01) contains one entry – the .01 field is a pointer to the DOMAIN file (#4.2). When VistALink is installed, the install process creates this entry and assigns the proper Domain Name using the Domain.

The site parameters in this top-level entry pertain to Foundations and VistALink. Currently, all the parameters in this file related to VistALink and listener configuration. As more Foundations tools are introduced, other non-VistALink related parameters would be added.

For each set of listeners that you plan to run on your system, you should make an entry in the VistALink LISTENER CONFIGURATION file and add listeners to the configuration. After adding a listener configuration, associate that configuration with any BOX-VOLUME where this configuration is appropriate.

As part of the VistALink installation for Cache systems, a listener configuration named 'DEFAULT' was created for port 8000.

4.2.7 Editing Site Parameters

To edit VistALink related site parameters, use the Site Parameters action.

```
HEARTBEAT RATE: 180// <Enter>
LATENCY DELTA: 180// <Enter>
```

Note: The following portion of the dialog is not presented to a DSM system user.

```
Select BOX-VOLUME PAIR: ROU:CACHE//
  BOX-VOLUME PAIR: ROU:CACHE// <Enter>
  DEFAULT CONFIGURATION: DEFAULT// <Enter>
Select BOX-VOLUME PAIR:
```

As part of this action, you are asked to select a Box-Volume Pair entry. Then, within each Box-Volume Pair entry (representing the volume set and system on which the listener should run), you can set the default listener configuration that should automatically be started as part of the execution of the XOBV LISTENER STARTUP option. Also, the Start Box action uses this default listener configuration.

Table 4, below, defines the Foundations site parameter fields.

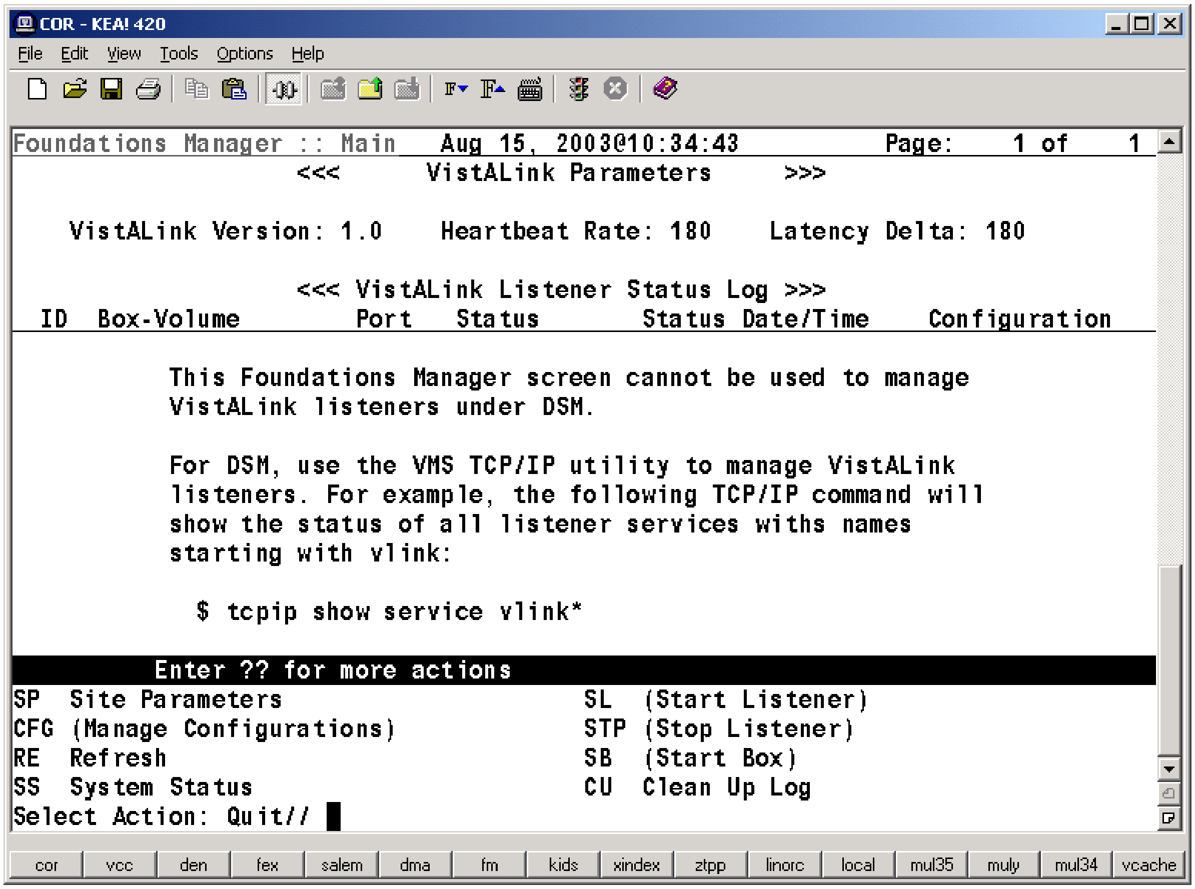
Field	Meaning
Heartbeat Rate	This field indicates the rate (in seconds) of the VistALink heartbeat message originating from a client. If there is no activity on the connection for this amount of time, the client will send a system heartbeat message. The client, as part of the initial connection protocol, retrieves this value. As a result, the client and the M server are always synchronized regarding the heartbeat rate.
Latency Delta	This field indicates the number of seconds to add to the HEARTBEAT RATE when calculating the initial timeout value for the VistALink listener. The client and the M server are synchronized regarding the HEARTBEAT RATE. This latency parameter allows the site to fine tune the timeout value. The site can to take into account any network slowness or other factors that may delay the arrival of the system heartbeat message from the client.
Box-Volume Pair	This field indicates the BOX-VOLUME pair for the entry. The XOBV LISTENER STARTUP option uses this field to find the configuration that should be used to startup VistALink listeners for the BOX-VOLUME pair. ***Note: This information is not presented to a DSM system user.***
Default Configuration	This field indicates the default startup listener configuration for the BOX-VOLUME PAIR entry. The XOBV LISTENER STARTUP option uses this field to retrieve the correct listener configuration from the VistALink LISTENER CONFIGURATIONS (#18.03) file. The information in the configuration is then used to startup the indicated VistALink listeners on the desired ports. ***Note: This information is not presented to a DSM system user.***

Table 4: Foundations/VistALink Site Parameter Entries for Cache Systems

4.3 Listener Management for DSM / VMS Systems

Access the VistALink system managers' option (XOBU SITE SETUP MENU) using the Operations Management menu, which is available via the Foundations Manager interface (**Figure 7**, below). This option is a ListMan application; for DSM / VMS systems, it includes the following executable actions:

```
SP   Site Parameters
RE   Refresh
SS   System Status
```


COR - KEA! 420
File Edit View Tools Options Help
cor vcc den fex salem dma fm kids xindex ztpp linorc local mul35 muly mul34 vcache

```
Foundations Manager :: Main   Aug 15, 2003@10:34:43          Page:     1 of     1
                         <<<      VistALink Parameters     >>>

    VistALink Version: 1.0    Heartbeat Rate: 180    Latency Delta: 180

                         <<< VistALink Listener Status Log >>>
  ID  Box-Volume        Port   Status        Status Date/Time     Configuration

           This Foundations Manager screen cannot be used to manage
           VistALink listeners under DSM.

           For DSM, use the VMS TCP/IP utility to manage VistALink
           listeners. For example, the following TCP/IP command will
           show the status of all listener services withs names
           starting with vlink:

             $ tcpip show service vlink*

          Enter ?? for more actions
SP  Site Parameters                        SL  (Start Listener)
CFG (Manage Configurations)                STP (Stop Listener)
RE  Refresh                                SB  (Start Box)
SS  System Status                          CU  Clean Up Log
Select Action: Quit//
```

Figure 7: Foundations Manager Interface – DSM / VMS Systems

Note: Parentheses around a displayed executable action [e.g. (Start Listener)] indicate that the action is disabled, or unavailable to the user.

Using DSM for OpenVMS, multi-threaded listeners are externally implemented through the TCP/IP Service for OpenVMS (previously known as UCX). The TCP/IP Service permits multiple TCP/IP clients to connect and run as concurrent processes, up to the limits established by the system. TCP/IP listens on a particular port, and launches a specified VistALink handler process for each client connection.

For the TCP/IP VistALink handler process, you need to create:

- An OpenVMS account
- A home directory
- A DCL (Digital Command Language) login command procedure

An example is used throughout this section. For this example the following names are used:

- The OpenVMS VistALink handler account name for the TCP/IP Service is VLINK
- The home directory is [VLINK]
- The DCL login command procedure is named VLINK.COM

4.3.1 Setting Up An OpenVMS User Account

The easiest way to configure an OpenVMS account to be a VistALink handler is to copy most of the parameters from VA MailMan TCP account. To do this:

1. Determine an unused User Identification Code (UIC), typically in the same group as other DSM for OpenVMS accounts.
2. Using the OpenVMS Authorize utility, copy the XMINET account to a new VLINK account with the unused UIC. You must have SYSPRV to do this.

Make sure that the account settings for the new VLINK account are the same as in the following example, or, if they are different, that the impact of the different settings is acceptable for your system. In particular, make sure that the DisCtlY, Restricted and Captive flags are set for security reasons.

4.3.2 Setting Up A Home Directory for the VistALink Handler Account

You need to create a home directory for the VistALink handler account. This directory will house the DCL command procedure that is executed whenever a client connects, as well as log files. Make sure that the owner of the directory is the VLINK account.

For example, to create a home directory named [VLINK] with ownership of VLINK:

```
$ CREATE/DIR [VLINK]/OWNER=VLINK
```

Creating a DCL Login Command Procedure for the VistALink Handler

Create a DCL command procedure in the home directory for the handler account. Make sure the command procedure file is owned by the VistALink handler account.

1. Adjust the DSM command line (environment, UCI and volume set) for your system.
2. If access control is enabled, ensure that the VLINK account has access to this UCI, volume set and routine (see "Access Control List (ACL) Issues", later in this chapter).

Sample DCL Login Command Procedure

```
$!VLINK.COM - for incoming connect requests
$!-------------------------------------------------------------
$set noon        !Don't stop
$ set noverify    !change as needed
$! set verify     !change as needed
$! WAIT 00:00:00
$ purge/keep=10 sys$login:VLINK*.log !Purge log files only
$! set proc/priv=(share)  !May not be required for MBX device
$  x=f$trnlnm("sys$net")     !This is our MBX device
$
$ write sys$output "Opening "+x !This can be viewed in the log file
$! Check status of the BG device before going to DSM
$ cnt=0
$ CHECK:
$ stat=f$getdvi("''x'","STS")
$ if cnt .eq. 10
$ then
$ write sys$output "Could not open "+ x
$ goto EXIT
$ else
$       if stat .ne. 16
$       then
$       cnt = cnt + 1
$       write sys$output "''cnt'> ''x' not ready!"
$       wait 00:00:01 !Wait one second to assure connection
$       goto CHECK
$       else
$       SET NOVERIFY
$!--------------------------------------------------------------------
$       dsm/env=DSMMANAG /uci=VAH /vol=ROU UCX^XOBVTCP
$!--------------------------------------------------------------------
$       endif
$ endif
$ EXIT:
$ logout/FULL
```

Note: In this sample procedure, 'environ,' 'uci' and 'vol' are site-specific.

> ***It is possible to run different TCP/IP Service listener processes on multiple nodes. It is not necessary to do this; however, depending on your site configuration and needs, you may find a need to do so.***

The steps to set up a TCP/IP Service for VistALink are:

1. Determine the part
2. Set up the "VLINK" TCP/IP Service
3. Enable and save the "VLINK" TCP/IP Service

4.3.3 Setting Up and Enabling the TCP/IP Service

Once you create the VistALink handler, create the TCP/IP Service to listen for connections and launch the VistALink handler. You need to choose:

- The OpenVMS node to run the listener on. Choose the node that you want to run the resulting M jobs on to process incoming VistALink messages. This is also the node whose IP address will be advertised to other systems as the location of your VistALink listener.
- The port it should listen on.
- The user account and command file name to invoke when a connection is received.

Prior to setting TCP/IP up in production, you can set up a "test" TCP/IP Service that logs into an M test account for testing. The "test" TCP/IP Service can use the same OpenVMS account and directory as the production TCP/IP Service. Just create a different DCL command file with the UCI and volume set of the test account.

4.3.3.1 Obtaining an Available Listener Port (for Alpha / VMS systems only)

Port selections conflict only if another process on the same system is using the same port. To list the ports currently in use on OpenVMS systems, use the DCL command:

```
$  TCPIP SHOW DEVICE_SOCKET
                                     Port                  Remote
Device_socket  Type     Local  Remote  Service           Host

  bg3         STREAM    8001       0  VistALink         0.0.0.0
  bg23        STREAM    9700       0  Z3ZTEST           0.0.0.0
  bg24        STREAM    9600       0  ZSDPROTO          0.0.0.0
```

For example, if 8000 shows up in the Local Port column, some other application is already using this port number; choose another port.

4.3.4 Creating the Service

Since the TCP/IP Service is node specific, make sure you are on the same node that you want the listener to run on.

```
$TCPIP
TCPIP> SET SERVICE VLINK/USER=VLINK/PROC=VLINK /PORT=8000-
_TCPIP> /PROTOCOL=TCP/REJECT=MESSAGE="All channels busy" -
_TCPIP> /LIMIT=50/FILE=SYS$SYSDEVICE:[VLINK]VLINK.COM

TCPIP> SHO SERVICE VLINK/FULL

Service: VLINK
                            State:     Disabled
Port:             8000      Protocol:  TCP             Address:  0.0.0.0
                            User_name: not defined     Process:  VLINK
```

Enabling and Saving the Service

Since TCP/IP is node specific, make sure you are on the same node that you want the listener to run on.

```
TCPIP> ENABLE SERVICE VLINK          (enable service immediately)
TCPIP> SET CONFIG ENABLE SERVICE VLINK  (save service for reboot)
TCPIP> SHO SERVICE/FULL VLINK

Service: VLINK
                            State:     Enabled
Port:             8000      Protocol:  TCP             Address:  0.0.0.0
Inactivity:          5      User_name: VLINK         Process:  VLINK
Limit:              50      Active:      0             Peak:       0

File:         SYS$SYSDEVICE:[VLINK]VLINK.COM
Flags:        Listen

Socket Opts:  Rcheck Scheck
 Receive:            0      Send:               0

Log Opts:     None
 File:        not defined

Security
 Reject msg:  All channels busy

 Accept host: 0.0.0.0
 Accept netw: 0.0.0.0
TCPIP> SHO CONFIG ENABLE SERVICE

Enable service
     FTP, FTP_CLIENT, VLINK, MPI, TELNET, XMINETMM
TCPIP> EXIT
```

Note: To test the connection, refer to Chapter 3, "Testing the Listener."

4.3.5 Access Control List (ACL) Issues

Some sites use DSM's ACL feature, which controls access explicitly to each OpenVMS account that needs to enter that DSM environment. If your site is using ACL, you should set up the VLINK account with PROGRAMMER access, and then specify the Volume set and UCI name that the VLINK user account has authorization to access. Ensure that the OpenVMS VLINK account prohibits Batch, Local, Dialup and Remote logins, allowing only Network logins.

An example of setting this level of access for an VLINK account is provided below:

```
$ DSM /MAN ^ACL

Environment Access Utilities

    1.  ADD/MODIFY USER                       (ADD^ACL)
    2.  DELETE USER                           (DELETE^ACL)
    3.  MODIFY ACTIVE AUTHORIZATIONS          (^ACLSET)
    4.  PRINT AUTHORIZED USERS                (PRINT^ACL)

Select Option > 1  ADD/MODIFY USER

OpenVMS User Name:   >   VLINK

ACCESS MODE    VOL       UCI       ROUTINE
-----------    ---       ---       -------

No access rights for this user.

Access Mode ([M]ANAGER, [P]ROGRAMMER, or [A]PPLICATION):   >    P
Volume set name:   >   VAH
UCI:   >   ROU
UCI:   >   <RET>
Volume set name:   >   <RET>
Access Mode ([M]ANAGER, [P]ROGRAMMER, or [A]PPLICATION):   >    <RET>

USER           ACCESS MODE    VOL       UCI       ROUTINE
----           -----------    ---       ---       -------

VLINK        PROGRAMMER     ROU       VAH

OK to file?   <Y>   <RET>

OpenVMS User Name:   >   <RET>

OK to activate changes now?   <Y>   <RET>

Creating access authorization file:  SYS$SYSDEVICE:[DSMMGR]DSM$ACCESS.DAT.
```

Example (Contains Recommended Settings)

1. Review the XMINET (TCP/IP MailMan) VMS account.

```
$SET DEF SYS$SYSTEM
$MC AUTHORIZE
UAF> SHOW XMINET

Username: XMINET                       Owner:  DSM
Account:                                   UIC:    [50,44] ([XMINET])
CLI:      DCL                              Tables: DCLTABLES
Default:  SYS$SYSDEVICE:[XMINET]
LGICMD:   NL:
Flags:  DisCtlY Restricted Captive
Primary days:   Mon Tue Wed Thu Fri
Secondary days:                     Sat Sun
Primary   000000000011111111112222  Secondary 000000000011111111112222
Day Hours 012345678901234567890123  Day Hours 012345678901234567890123
Network:  ##### Full access ######            ##### Full access ######
Batch:    -----  No access  ------            -----  No access  ------
Local:    -----  No access  ------            -----  No access  ------
Dialup:   -----  No access  ------            -----  No access  ------
Remote:   -----  No access  ------            -----  No access  ------
Expiration:   (none)    Pwdminimum:  6   Login Fails:     0
Pwdlifetime:  90 00:00    Pwdchange:      (pre-expired)
Last Login:   (none) (interactive), 10-FEB-1998 15:30 (non-interactive)
Maxjobs:         0  Fillm:       500  Bytlm:       100000
Maxacctjobs:     0  Shrfillm:      0  Pbytlm:           0
Maxdetach:       0  BIOlm:       150  JTquota:       4096
Prclm:           8  DIOlm:        18  WSdef:         1344
Prio:            4  ASTlm:       176  WSquo:         2688
Queprio:         4  TQElm:        10  WSextent:     65536
CPU:        (none)  Enqlm:      3000  Pgflquo:     100000
Authorized Privileges:
  NETMBX    OPER      SHARE     TMPMBX
Default Privileges:
NETMBX     OPER       SHARE      TMPMBX
```

2. Copy XMINET (TCP/IP MailMan) account to a new account with unused UIC

Note: This example assumes that UIC [51,45] is an unused UIC. Substitute an unused UIC on your system.

```
UAF> COPY /ADD XMINET VLINK/UIC=[51,45]
%UAF-I-COPMSG, user record copied
%UAF-W-DEFPWD, copied or renamed records must receive new password
%UAF-I-RDBADDMSGU, identifier VLINK value [000051,000045] added to rights
database

UAF> SHOW VLINK

Username: VLINK                          Owner:  DSM
Account:                                   UIC:    [51,45] ([VLINK])
CLI:      DCL                              Tables: DCLTABLES
Default:  SYS$SYSDEVICE:[XMINET]
LGICMD:   NL:
Flags:  DisCtlY Restricted Captive
Primary days:   Mon Tue Wed Thu Fri
Secondary days:                     Sat Sun
Primary   000000000011111111112222  Secondary 000000000011111111112222
```

```
Day Hours 012345678901234567890123  Day Hours 012345678901234567890123
Network:  ##### Full access ######            ##### Full access ######
Batch:    -----  No access  ------            -----  No access  ------
Local:    -----  No access  ------            -----  No access  ------
Dialup:   -----  No access  ------            -----  No access  ------
Remote:   -----  No access  ------            -----  No access  ------
Expiration:    (none)    Pwdminimum:  6   Login Fails:     0
Pwdlifetime:   90 00:00    Pwdchange:      (pre-expired)
Last Login:    (none) (interactive),            (none) (non-interactive)
Maxjobs:         0  Fillm:       500  Bytlm:       100000
Maxacctjobs:     0  Shrfillm:      0  Pbytlm:           0
Maxdetach:       0  BIOlm:       150  JTquota:       4096
Prclm:           8  DIOlm:        18  WSdef:         1344
Prio:            4  ASTlm:       176  WSquo:         2688
Queprio:         4  TQElm:        10  WSextent:     65536
CPU:        (none)  Enqlm:      3000  Pgflquo:     100000
Authorized Privileges:
  NETMBX    OPER      SHARE     TMPMBX
Default Privileges:
  NETMBX    OPER      SHARE     TMPMBX
```

3. Modify home login directory of the new account.

```
UAF> MOD VLINK/DIR=[VLINK]
%UAF-I-MDFYMSG, user record(s) updated
UAF> EXIT

%UAF-I-DONEMSG, system authorization file modified
%UAF-I-RDBDONEMSG, rights database modified
```

4.3.6 How to Control the Number of Log Files Created by the TCP/IP Service

The VLINK TCP/IP Service automatically creates log files (this cannot be prevented) in the VLINK directory named “VLINK.LOG;xxx,” where '**xxx**' is a file version number. New versions of this file will be created until that file version number reaches the maximum number of 32767. In order to minimize the number of log files created, you may want to initially rename this log file to the highest version number with the command:

```
$ RENAME disk$:[VLINK]VLINK.LOG; disk$:[VLINK]VLINK.LOG;32767
```

Alternatively, you can set a limit on the number of versions of the log file that can concurrently exist in the VLINK directory:

```
$ SET FILE /VERSION_LIMIT=10 disk$:[VLINK]VLINK.LOG;
```

Note: You probably should not limit the number of versions of the log file until you know your VistALink service is working correctly; keeping the log files can help when diagnosing problems with the service/account.

4.3.7 Editing the Foundations / VistALink Site Parameters for OpenVMS

To edit VistALink related site parameters, use the Site Parameters action on the Foundations Manager interface.

```
HEARTBEAT RATE: 600// <Enter>
LATENCY DELTA: 180// <Enter>
```

Definitions for the OpenVMS site parameter fields appear in **Table 5**, below.

Field	Meaning
Heartbeat Rate	This field indicates the rate the VistALink heartbeat message should be expected from a client. If there is no activity on the connection for this amount of time, the client will send a system heartbeat message. The client, as part of the initial connection protocol, retrieves this value. As a result, the client and the M server are always synchronized regarding the heartbeat rate.
Latency Delta	This field indicates the number of seconds to add to the HEARTBEAT RATE when calculating the initial timeout value for the VistALink listener. The client and the M server are synchronized regarding the HEARTBEAT RATE. This latency parameter allows the site to fine tune the timeout value. The site can to take into account any network slowness or other factors that may delay the arrival of the system heartbeat message from the client.

Table 5: Foundations/VistALink Site Parameter Entries – Cache Systems Description(s)

Chapter 5: Java Logging Management

5.1 Using Loggers

Logging is an important feature of any product as proper logging implementation allows shorter troubleshooting times in production environment. VistALink uses log4j http://jakarta.apache.org/log4j/docs/index.html logging framework for both debug and error logging on the client side.

VistALink logging can be enabled on the end-user client workstation for any client application using the VistALink libraries. To enable logging for VistALink, the Log4J library (log4j-1.x.x.jar - version 1.2.7 or higher) needs to be installed on the client workstation.

For log4j to work properly it has to be configured. If log4j configuration is not provided, you will see a few warnings in the system output console about log4j initialization problems and no information will be logged from VistALink.

Here is a sample log4j configuration file.

log4jConfig.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
  <!--
          Very verbose console appender.
          Prints out among other things class name, method name and
          line number where logging is coming from.
          Caution: this will slow down your system considerably, so use it
          only for development and testing purposes.
   -->
  <appender name="verboseConsoleAppender" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
       <param name="ConversionPattern" value="%-4r [%t] %-5p class %C method %M
line number %L category %c %x - %m%n"/>
    </layout>
  </appender>

  <!--
          Detail console appender prints out less info than verbose console
appender
          and does not produce as much performance impact as verbose appender.
          Can be used in production environment.
   -->
  <appender name="detailConsoleAppender" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
       <param name="ConversionPattern" value="%-4r [%t] %-5p %x %m%n"/>
    </layout>
  </appender>

  <!--
          Shortest and least useful console appender as it will not print out any
          additional info other than the message being passed into the logging
method.
   -->
  <appender name="briefConsoleAppender" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
       <param name="ConversionPattern" value="%m%n"/>
    </layout>
  </appender>


  <!--
          Audit console appender that is used to capture VistALink interaction
time.
          Use org.apache.log4j.RollingFileAppender
          instead of org.apache.log4j.ConsoleAppender to send output to the file.
   -->
  <appender name="interactionAuditConsoleAppender"
class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
       <param name="ConversionPattern" value="Time to execute VistALink
interaction: %m%n"/>
    </layout>
  </appender>

  <!--
          VistaSocketConnection interaction execution audit timer logger.
          Do not disable this as you will loose info on how much time it takes
          to execute Java to M interactions.
       -->
  <logger name="gov.va.med.foundations.adapter.spi.VistaSocketConnection.AuditLog"
additivity="false">
        <level value="info" />
       <appender-ref ref="interactionAuditConsoleAppender"/>
  </logger>

  <logger name="gov.va.med.foundations.samples.VistaLinkRpcSample"
additivity="false">
        <level value="debug" />
       <appender-ref ref="briefConsoleAppender"/>
  </logger>

  <logger name="gov.va.med.foundations.samples.VistaLinkRpcSample.Other"
additivity="false">
        <level value="debug" />
       <appender-ref ref="briefConsoleAppender"/>
  </logger>

  <logger name="gov.va.med.foundations" additivity="false" >
        <level value="debug" />
       <appender-ref ref="detailConsoleAppender"/>
  </logger>

  <root>
    <priority value ="debug" />
    <appender-ref ref="detailConsoleAppender"/>
  </root>

</log4j:configuration>
```

As you can see, this file defines four console appenders: *verboseConsoleAppender*, *detailConsoleAppender*, *briefConsoleAppender* and *interactionAuditConsoleAppender* that use different output patterns. To see other possible pattern parameters see:

http://jakarta.apache.org/log4j/docs/api/org/apache/log4j/PatternLayout.html.

Note that some parameters might slow down performance of your system considerably, especially *%C, %M, %L*. This example uses console appender; other appenders such as *RollingFileAppender* can be used.

VistALink uses its fully qualified class names for logger names. The log4j config file on the previous page shows five loggers:

1. *root*
2. *gov.va.med.foundations*
3. *gov.va.med.foundations.adapter.spi.VistaSocketConnection.AuditLog*
4. *gov.va.med.foundations.samples.VistaLinkRpcSample*
5. *gov.va.med.foundations.samples.VistaLinkRpcSample.Other*.

The super parent of all loggers is the root logger. If logger is not specified, the next available logger will be used. So for example if *gov.va.med.foundations.utilities.AuditTimer* logger is not specified in the log4j config file, then log4j framework will look for logger named *gov.va.med.foundations.utilities*. If this logger is not specified in the log4j config file, then log4j framework will look for the logger named gov.va.med.foundations logger. If no loggers are found, the root logger will be used. This allows for highly granular logging control.

Log level is another important parameter that can be controlled by the log4j configuration file. *<level value="debug" />* specifies which log level should be printed. Use *debug* level when you need to see the details of VistALink logging. Use *error* when you need to see only errors being printed. Use *off* level to turn logging off for all levels pertaining to a specific logger.

5.1.1 Recommended Loggers

gov.va.med.foundations – use this logger to control all Foundations logging.

gov.va.med.foundations.adapter.spi.VistaSocketConnection.AuditLog – enable this logger to see performance statistics for the VistALink to M interactions. Output can be redirected to a separate file using a separate appender.

5.1.2 Specifying the log4j Configuration File

There are a few ways to specify log4j configuration file at runtime.

1. *Current directory* – For quick test purposes, you can put the log4j configuration file into the working directory where you are running your application. Since in Java the current directory is NOT the directory of the Java application, this could get tricky.
2. *Command line parameters* – Use *-Dlog4j.configuration=props/log4jConfig.xml* to pass in the location of the log4j configuration file. **Important**: this location is NOT a location relative to the current directory, but instead a location relative to your Java *CLASSPATH*. This is a very important distinction. So, in a Web application environment you could copy your *props/log4Config.xml* file to the *yourWebAppDir/WEB-INF/classes* directory, as this directory is in your classpath by default.

3. *Use the log4j API from your code to load the configuration file relative to the current directory* - use *DOMConfigurator.configure("props/log4jConfig.xml");* in your application code to specify the log4j configuration file. Location of the config file is **not** the same as when passing the config file in the command line parameter; here the config file location is relative to the current directory.
4. *Use log4j API from your code to load configuration file from classpath.* In this case one can retrieve configuration file as a resource URL using *MyClass.class.getClassLoader().getResource("props/log4jConfig.xml")* and pass it into *DOMConfigurator.configure(log4jConfigResourceURL)*. In this case location of the config file **is** the same as when passing config file in command line parameter - config file location is relative to the *CLASSPATH*.

5.1.3 Preventing Users From Snooping log4j Logs in J2SE Applications

With the log4j framework used in the J2SE environment it is important to understand the importance of not letting users specify their own log 4j configuration files. This would become a security issue if users were able to see all VistALink security traffic being logged.

Application developers can prevent this by making sure to use log4j configuration APIs. (*DOMConfigurator.configure("props/log4jConfig.xml");*) to specify the log4j config file that resides in a digitally signed jar files. Use log4j configuration APIs will override any command line parameters that users might be trying to pass in, hence no other log4j config file could be used. Digitally signing your jar file with the log4j config file inside it prevents users from modifying the contents of this file.

Chapter 6: Security Management

The subsections in this chapter detail VistALink security management.

6.1 Authentication Security

VistALink authentication, like that of the RPC Broker, is a wrapper around the Kernel login on your M system. It obeys the same authentication rules as other Kernel logins on your system, and uses Kernel code to obtain a "yes/no" authentication decision for each login attempt. As such, no new "user management" tools are required, other than the standard Kernel user management system.

The VistALink M listener process receives and processes connection requests from client applications. Connection by those client applications is subject to Kernel authentication as any normal login requires.

To authenticate incoming connections to M, VistALink wraps Kernel's current authentication mechanism. To authorize incoming RPC requests, VistALink wraps the RPC Broker's current RPC authorization mechanism.

Security with the RPC Broker is a four-part process:

1. Client workstations must send a valid connection request to the M Server.
2. Users must have valid signon credentials recognized by Kernel (typically access and verify codes).
3. Users must be authorized users (on the M system) of the application whose RPC calls are being invoked by the client application.
4. Any RPC must be registered and valid for the application being executed.

6.2 Authentication Timeout Behavior

Prior to successful login, the VistALink login timeout on the client is set to the value of the *Heartbeat Rate* as set in the VistALink site parameters for your site. The VistALink login dialogs time out based on this value, dropping the (unauthenticated) connection to VistA.

After successful login, the M server side timeout is set by Kernel (in the variable DTIME). This value is also returned for application use on the client side as part of the result of a successful login. However, it is up to the application how it decides to implement client-side timeouts.

6.3 RPC Authorization

Client applications can use any RPC Broker remote procedure call (RPC) authorized to a Kernel “B”-type option, if the “B”-type option is authorized to the signed-on user. Through this mechanism, data is typically exchanged between clients and the VistALink server.

For more information regarding security enforced on RPCs, please see the *RPC Broker Systems Manual,* available at http://www.va.gov/vdl/.

6.4 Limited Kernel Auto-Signon Support

VistALink supports Kernel auto-signon with the following restrictions (in addition to the existing requirements for Kernel auto-signon):

- The Broker client agent must already be running on the workstation.
- Either an RPC Broker, Telnet, or VistALink connection may be the first active connection.
- On DSM systems, Kernel auto-signon for VistALink application is not currently supported, but may be in the future. The current lack of support is related to differences in the way VistALink processes are started and how client IP addresses are retrieved on DSM systems.

Finally, future implementation of other SSO solutions may result in the deprecation of Kernel auto-signon.

6.5 Logger Security

VistALink uses the Log4J logging utility to write information to a configurable log. The information written should not be security-sensitive. However, you may wish to disable VistALink's logging output. For information on the security implications of VistALink's client-side Log4J logging, please see the previous chapter on VistALink Logging.

Part III – VistALink Programming

Chapter 7: Authenticating and Connecting to VistA

7.1 Overview

The J2SE version of VistALink establishes a connection from the Java client to the M server. This connection remains open until closed by the client (unless the server is shut down).

The high-level steps to establish a VistALink connection to M are:

1. Providing server configuration information to VistALink (IP address and port of the M VistALink listener to connect to)
2. Authenticating the end-user over the connection
3. Executing RPCs
4. Closing the connection (logging out)

VistALink uses the Java Authentication and Authorization Service (JAAS) framework for steps 1, 2 and 4 above (providing server configuration to VistALink; authenticating the end-user; and logging out). For information on step 3, executing RPCs, see the Chapter 8 in this manual, “Executing Requests.”

7.1.1 JAAS Overview

JAAS is a java pluggable framework for user authentication and authorization. “Pluggability” means different security modules (e.g., authentication modules) can be added or “plugged in” to an application without recompiling the application. VistALink uses the JAAS framework to authenticate end-users to an M/Kernel system, via the users' customary Kernel access and verify codes.

A JAAS-compliant login module contains all of the logic required to authenticate a user to a given system. The login module class does not itself, however, include the user interface to gather authentication credentials (e.g., access and verify codes) from the end-user. Instead, a set of JAAS-compliant callbacks, along with a JAAS-compliant callback handler, are used to de-couple the user interface from the login module. VistALink provides a JAAS-compatible login module, and JAAS-compliant callbacks and callback handlers, to perform a VistA login.

The JAAS framework also provides authorization capabilities; VistALink, however, uses JAAS for authentication only. VistALink does not make any use of the permission/authorization portions of the JAAS specification at this time.

7.2 VistALink JAAS Implementation

7.2.1 VistaLoginModule

VistALink provides a single JAAS-compliant login module class, VistaLoginModule. As a developer, you do not use this class directly; instead, your application:

- Specifies which login module to use, via a JAAS configuration file
- Creates a LoginContext instance, and passes it a supported callback handler instance to collect user input
- Invokes the login method of the LoginContext class to initiate the login process for the configured login module

7.2.2 JAAS Login Configuration Overview

By default, VistALink uses the default JAAS configuration reader to load login configurations. The default JAAS configuration reader class loads login configurations from a JAAS configuration file, which it expects to be in a predefined format.

One or more configuration entries are defined in the JAAS configuration file. The configuration file itself can have any name, and can be located anywhere. Each entry in the JAAS configuration file defines a particular login configuration. Generically, the format of this file is:

```
ConfigurationName {
            ModuleClass Flag ModuleOptions;
      };

      ConfigurationName {
            ModuleClass Flag ModuleOptions;
      };
```

7.2.3 VistALink-Specific JAAS Login Configuration

Specifically for VistALink, an example of the format of the needed JAAS configuration file is:

```
 Test {
   gov.va.med.foundations.security.vistalink.VistaLoginModule requisite
   gov.va.med.foundations.security.vistalink.ServerAddressKey="10.21.185"
   gov.va.med.foundations.security.vistalink.ServerPortKey="18010";
};
Production {
   gov.va.med.foundations.security.vistalink.VistaLoginModule requisite
   gov.va.med.foundations.security.vistalink.ServerAddressKey="10.21.1.85"
   gov.va.med.foundations.security.vistalink.ServerPortKey="8005";
};
```

This example defines two login configurations, one named "Test" and one named "Production." An application uses this name ('Test' or 'Production') as the index to retrieve a particular configuration from the JAAS configuration file.

To configure a VistALink login to a VistA system, configure a single login module per login configuration entry, within each entry's {braces}, as follows:

1. Name the VistALink login module class, including package name: gov.va.med.foundations.security.vistalink.VistaLoginModule
2. Follow with a flag indicating what action to take if login fails; for VistALink, use: *requisite*
3. Follow with options for the VistALink login module; there are two options that must be set, in "name=value" format:
   - gov.va.med.foundations.security.vistalink.ServerAddressKey
   - gov.va.med.foundations.security.vistalink.ServerPortKey.

   Use quotes around the server address and server port values.
4. Before the closing brace, end with a semicolon.
5. Follow the closing brace with a semicolon.

For more information about the JAAS configuration file format expected by the default JAAS configuration file reader class, see:

http://java.sun.com/j2se/1.4.1/docs/guide/security/jgss/tutorials/LoginConfigFile.html.

Note: It is possible to define your own JAAS configuration reader class instead of using the default class. If you do this, you are still responsible for providing the package/name of the VistaLoginModule class, the JAAS "requisite" flag, and the two options required by the VistaLoginModule.

7.2.4 Passing the JAAS Login Configuration(s) to Your JVM

The JAAS Login Configuration needs to be passed to the JVM (and hence to your application). The JAAS configuration can be passed in two ways:

- The javax.security.auth.login.Configuration java virtual machine (JVM) argument when launching your application, e.g.,

  ```
  java  -Djava.security.auth.login.config=jaas.config  MyApp
  ```

- In the Java security properties file.

In most cases it is preferable to use the JVM argument, since this allows the setting to be application-specific rather than machine-wide.

7.2.5 Selecting the JAAS Configuration From an Application

Once your application is running, it should select a specific configuration. In order to allow local administration of JAAS configuration files, you should in most cases provide a command-line parameter to allow local administrators to pass a particular JAAS configuration into your application. For example:

```
java -Djava.security.auth.login.config=jaas.conf  MyApp  Production
```

7.2.6 VistaLoginModule Callback Handlers

In order to de-couple the user interface for logon from the login module, the JAAS standard allows login modules such as VistaLoginModule to supply different callback handlers. VistALink supplies two callback handler classes, one for an interactive logon, and one for non-interactive unit testing:

- **CallbackHandlerSwing**: for production application use. Collects access code, verify code, division and "change verify code" input via a set of Swing dialogs.
- **CallbackHandlerUnitTest**: for unit testing only (not production use). Access code, verify code, division are passed as parameters to the class constructor, resulting in a "silent" login suitable for (non-interactive) unit testing. "change verify code" functionality is not supported.

Part of the JAAS VistALink login involves instantiating one of these two callback handler classes and passing the class as a parameter to create a JAAS login context (see below).

7.3 Putting the Pieces Together: VistALink JAAS Login

7.3.1 Logging in to VistA

The following is an example login. If application execution succeeds through the try block, the user has successfully logged in to the specified VistA listener.

```
// variable holding LoginContext should have application scope
// since it will be needed to log out, later on


try {

    // create the callback handler to use to collect user input
    // pass current Frame as parameter
    CallbackHandlerSwing cbhSwing = new CallbackHandlerSwing(myFrame);

// create the LoginContext to control the login process;
    // pass the JAAS configuration to connect to, and the callback
    // handler (jaasConfigName value could be passed in from command
    // line)
    loginContext = new LoginContext(jaasConfigName, cbhSwing);

    // login to server through the LoginContext


} catch (VistaLoginModuleException e) {

    JOptionPane.showMessageDialog(null, e.getMessage(), "Login error",
      JOptionPane.ERROR_MESSAGE);

} catch (LoginException e) {

    JOptionPane.showMessageDialog(null, e.getMessage(), "Login error",
      JOptionPane.ERROR_MESSAGE);

}
```

7.4 After Successfully Logging In

7.4.1 Retrieving the VistaKernelPrincipal

Upon successful login, the JAAS subject (available from the JAAS LoginContext class after a successful login) contains a JAAS principal (user entity), which holds:

- Demographic information about the logged-in user
- The authenticated VistALink connection object

The following code shows how to retrieve the Kernel principal after a successful login:

```
// variable holding Kernel principal may need application scope
// since it will be needed for RPC execution
private VistaKernelPrincipalImpl userPrincipal = null;

// . . . login code

// get the Kernel principal after logon
try {
    userPrincipal = VistaKernelPrincipalImpl.getKernelPrincipal(
      loginContext.getSubject());
}  catch (FoundationsException e) {
    JOptionPane.showMessageDialog(null, e.getMessage(), "Login error",
       JOptionPane.ERROR_MESSAGE);
}
```

After login, it is conceivable (in the future) that *more* than one principal could be contained in the JAAS subject, if multiple login modules were used. This might happen if some kind of compound login has been configured requiring several logins to complete, e.g., one for a Kernel M system, and one for a separate health data repository. Only one *Kernel* principal should ever be returned, however. Use the getKernelPrincipal helper method in the VistaKernelPrincipalImpl class to retrieve the single Kernel principal.

7.4.2 Retrieving the Authenticated Connection From the Principal

To execute RPCs, you'll need to retrieve the authenticated connection. The authenticated connection object over which you make requests is stored in the Kernel principal, and can be retrieved with the getAuthenticatedConnection method.

Once a successful login has been completed, retrieve the associated authenticated connection from the Kernel principal. You can then use this connection – which is "logged in" to the M system under the end-user's identity – to execute requests such as RPCs on behalf of the end-user.

For more information on executing requests, see Chapter 8 of this manual, "Executing Requests." An example of successful login appears below.

```
VistaLinkConnection myConnection =
  userPrincipal.getAuthenticatedConnection();

// . . . now you can execute requests
```

For information on how to use the VistaLinkConnection object to execute requests, see the chapter on executing requests, below.

7.4.3 Retrieving User Demographic Information

Use the following predefined static KEY* strings to retrieve user demographic values via the Kernel principal's getUserDemographicValue method. For example:

```
// get the DUZ
String duz = this.userPrincipal.getUserDemographicValue(
  VistaKernelPrincipalImpl.KEY_DUZ);

// get the name
String name = userPrincipal.getUserDemographicValue(
  VistaKernelPrincipalImpl.KEY_NAME_DISPLAY);
```

The complete set of returned demographics information (and keys) is:

Key	**Value**
KEY_DIVISION_IEN	Login division station IEN
KEY_DIVISION_STATION_NAME	Login division station name
KEY_DIVISION_STATION_NUMBER	Login division station number
KEY_DTIME	user timeout value
KEY_DUZ	DUZ
KEY_LANGUAGE	User language
KEY_NAME_DEGREE	User degree
KEY_NAME_FAMILYLAST	Name component family-last
KEY_NAME_GIVENFIRST	Name component given-first
KEY_NAME_MIDDLE	Name component middle
KEY_NAME_NEWPERSON01	New Person .01 Field name
KEY_NAME_PREFIX	Name component prefix
KEY_NAME_SUFFIX	Name component suffix
KEY_SERVICE_SECTION	User service/section
KEY_NAME_DISPLAY	Concatenated standard name
KEY_TITLE	User title

7.4.4 Logging Out

Your application should *always* call the logout method of the JAAS LoginContext class to log out of VistA before exiting. This ensures that proper Kernel clean up (e.g., cleanup of the ^TMP global) occurs on the M server to which the user was connected.

Logging Out of Swing Applications

In a Swing application, your application should always call the LoginContext's logout method when the application is shut down. There are a number of ways an application can be shut down (the user closes the application window, the application is terminated from the Windows control panel, etc.).

A good way to catch all of these shutdown cases is to implement a WindowAdapter as a window listener in the application, and provide an implementation of its windowClosing method that calls the LoginContext's logout method.

For example:

```
// loginContext has been defined earlier, with application scope

// add event listener to log out when window closes
frame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        mylogout();
        System.exit(0);
    }
});

// Method called from event handler to perform logout
private void mylogout() {
    // Kernel logout
    if (this.userPrincipal != null) {

        try {
            loginContext.logout();
        } catch (LoginException e) {
            JOptionPane.showMessageDialog(null, e.getMessage(),
              "Logout error", JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

7.5 Catching Login Exceptions

The LoginContext login() and logout() methods only throw exceptions that derive from LoginException. So at a minimum, when executing the login or logout methods of a LoginContext object, your application needs a try/catch block to catch LoginException.

VistaLoginModule provides more granular exceptions derived from LoginException so that your application can optionally filter exceptions at a finer level of granularity, meaning that your application can detect and implement specific processing for login exception types that might be of interest.

7.5.1 VistaLoginModule Exceptions

Exception	**Description**
VistaLoginModuleException	Like a LoginException, but may contain nested exception(s) that were the cause for the LoginException.
VistaLoginModuleLoginsDisabledException	Logins are disabled on the M server.
VistaLoginModuleNoJobSlotsAvailableException	Job slot maximum has been exceeded on M server.
VistaLoginModuleNoPathToListenerException	No reachable listener was found on the path represented by the specified IP address and Port.
VistaLoginModuleTooManyInvalidAttemptsException	The user tried to login too many times with invalid credentials.
VistaLoginModuleUserCancelledException	The user cancelled the login.
VistaLoginModuleUserTimedOutException	The user timed out of the login.

For example, if your application is interested in whether the IP and port specified were "bad" (at least at the time the login was attempted), you can trap for the VistaLoginModuleNoPathToListener exception, in addition to the standard LoginException:

```
try {

    // create the callback handler to use to collect user input
    CallbackHandlerSwing cbhSwing = new CallbackHandlerSwing(myFrame);

    // create the LoginContext to control the login process.
    loginContext = new LoginContext(serverAlias, cbhSwing);

    // login to server through the LoginContext
    loginContext.login();
} catch (VistaLoginModuleLoginsDisabledException e) {
   JOptionPane.showMessageDialog(
           null,
           "Logins are disabled; try later.",
           "Login error",
           JOptionPane.ERROR_MESSAGE);

} catch (LoginException e) {

       JOptionPane.showMessageDialog(null, e.getMessage(), "Login error",
         JOptionPane.ERROR_MESSAGE);
```

7.6 Unit Testing and VistALink

Because of the pluggable JAAS architecture, the user interface for login (to collect information from the end-user such as username and password) is separate from the logic that implements login. The user interface is contained in a JAAS-compliant set of callbacks; the login logic is contained in a JAAS-compliant login module. Therefore, the JAAS framework makes it straightforward to implement alternative user interfaces for login.

VistALink provides an alternative callback handler that implements a "silent" login suitable for unit testing purposes (***but not suitable for any production environment***). Your application passes the access code, verify code (and optionally division) to use to silently (non-interactively) login your application. Changing verify code is not supported with this callback handler.

For example:

```
   // Connection info
   String cfgName = "Production";

   // signon credentials for unit test callback handler
   String accessCode = "asdf.123";
   String verifyCode = "asdf.456";
   String division = "";

   try {

       // create the "unit test" callbackhandler for JAAS login
       CallbackHandlerUnitTest cbhUnitTest =
         new CallbackHandlerUnitTest(accessCode, verifyCode, division);

       // create the JAAS LoginContext for login
       lc = new LoginContext(cfgName, cbhUnitTest);

       // login to server
       lc.login();

} catch (VistaLoginModuleException e) {

       JOptionPane.showMessageDialog(null, e.getMessage(), "Login error",
         JOptionPane.ERROR_MESSAGE);

   } catch (LoginException e) {

       JOptionPane.showMessageDialog(null, e.getMessage(), "Login error",
         JOptionPane.ERROR_MESSAGE);
```

Chapter 8: Executing Requests

8.1 Remote Procedure Calls (RPCs)

A remote procedure call (RPC) is a defined call to M code that runs on an M server. A client application, through the RPC Broker, can make a call to the M server and execute an RPC on the M server. This is the mechanism through which a client application can:

- Send data to an M server
- Execute code on an M server
- Retrieve data from an M server

An RPC can take optional parameters to do some task and then return either a single value or an array to the client application.

*****For detailed information on RPCs, please refer to Getting Started With the Broker Development Kit (BDK) and/or the RPC Broker Technical Manual. Find both these publications at http://www.va.gov/vdl/. *****

8.2 Request Processing

During interactions with M from Java, developers use the RpcRequest object. The RpcRequest object encapsulates the data that will be sent to M to execute an interaction, i.e. RPC name, RPC parameters and RPC context. The RpcRequest object is constructed using the RpcRequestFactory object. The RPC parameters are accessed through the RpcRequest objects via clearParams(), getParams() and setParams() methods.

8.2.1 Get an RpcRequest Object: RpcRequestFactory Class

The *RpcRequest* class represents a request from Java to M. As the transport format, it permits the use of XML or a proprietary format that is faster for large amounts of data. Also, this class exposes methods for specifying Rpc Name, Rpc Context and the parameters used by M to execute the RPC.

The *RpcRequestFactory* class is responsible for creating instances of RpcRequest. In order to create an RpcRequest, the developer must call the static getRpcRequest method on this class. In the example shown below, getRpcRequest is overloaded with three declarations.

```
public static RpcRequest getRpcRequest() throws FoundationsException
```

This method is used to create a default RpcRequest with no specified Rpc Name or Rpc Context. You must specify the Rpc Context and the Rpc Name on the RpcRequest object before you can use this object in an interaction. Refer to javadoc on RpcRequest for more information.

```
public static RpcRequest getRpcRequest(String rpcContext) throws
FoundationsException
```

This method is used to create a RpcRequest with the specified Rpc Context. You must specify the Rpc Name on the RpcRequest object before you can use this object in an interaction. Refer to the javadoc on RpcRequest for more information.

```
public static RpcRequest getRpcRequest(String rpcContext, String
rpcName) throws FoundationsException
```

This method is used to create a RpcRequest with the specified Rpc Context and the Rpc Name. You may still specify another Rpc Context and Rpc Name on this object. Refer to the javadoc on RpcRequest for more information.

```
J2SE Example:
RpcRequest vReq = null;

//The Rpc Context
String rpcContext = "XOBV VISTALINK TESTER";

//The Rpc to call
String rpcName = "XWB GET VARIABLE VALUE";

//Construct the request object
try{
   vReq = RpcRequestFactory.getRpcRequest(rpcContext, rpcName);
}catch(FoundationsException e){
   // process exception as needed
}
```

8.2.2 Set RpcRequest Parameters: Explicit Style

There are two ways of passing RPC parameters to an RpcRequest object. For the sake of convenience, the first method is referred to as *explicit style*, the second method as *list style*.

In *explicit* style, the method of passing RPC parameters corresponds very closely to the underlying RPC parameters. The RPC Broker has three input parameter types for RPC calls. These map to VistALink RpcRequestParam parameter types as follows:

RPC parameter type	VistALink RpcRequestParam type
Literal	"string"
Reference	"ref"
List	"array"

To pass parameters into an RpcRequest, retrieve the RpcRequest's mutable RpcRequestParam object, accessible by RpcRequest.getParams(). Then, on that object, call setParam() to define each RPC parameter:

void setParam(int position, String type, Object value)

The parameters to this call are:

- position: the expected RPC parameter list position where the RPC expects to see the RPC parameter
- type: can be "string", "ref" or "array" (to match the expected RPC parameter type)
- value: an object that is the value of the RPC parameter. For "string" or "ref" types, the object should be a string. For "array" types, the object should implement the Map, List or Set interface.

Literal RPC Parameter Example:

```
RpcRequest vReq = RpcRequestFactory.getRpcRequest();
vReq.getParams().setParam(1, "string", "I am a string");
```

Reference RPC Parameter Example:

```
RpcRequest vReq = RpcRequestFactory.getRpcRequest();
vReq.getParams().setParam(1, "ref", "DTIME");
```

List RPC Parameter Example:

```
RpcRequest vReq = RpcRequestFactory.getRpcRequest();
ArrayList nums = new ArrayList();
nums.add("3");
nums.add("5");
nums.add("4");
nums.add("1");
nums.add("7");
nums.add("8");
nums.add("2");
nums.add("6");
nums.add("9");
vReq.getParams().setParam(1, "array", nums);
```

Combination RPC Parameter Example:

```
RpcRequest vReq = RpcRequestFactory.getRpcRequest();
ArrayList nums = new ArrayList();
nums.add("3");
nums.add("5");
nums.add("4");
nums.add("1");
nums.add("7");
nums.add("8");
nums.add("2");
nums.add("6");
nums.add("9");
vReq.getParams().setParam(1, "array", nums);
vReq.getParams().setParam(2, "string", "I am a string");
vReq.getParams().setParam(3, "ref", "DTIME");
```

8.2.3 Set RpcRequest Parameters: List Style

A second style of passing RPC parameters for an RPC into an RpcRequest object is called the *List* style. This method offers a small amount of abstraction away from the underlying RPC, although parameters still must correspond directly to what is expected by the RPC being invoked.

With List style, you create an object that implements the List interface, and that holds each of the parameters as an object entry in the list. Add each parameter as an object value to the List, and then use RpcRequest.setParams(List) to pass the RPC parameters to the request..

The RpcRequest object processes the List internally as follows, to extract the RPC parameter characteristics for the request:

- position: Determined by the order each object was added to the List. The first object added becomes the first RPC parameter, the second becomes the second, and so forth.
- type:
  - If an object found in the List is a String, it is passed as an RPC literal parameter.
  - If an object found in the List implements the Map, List, or Set interfaces, it is passed as an RPC List parameter.
  - If an object found in the List is an instance of the special RpcReferenceType class, it is passed as an RPC reference parameter.
- value: The value for the RPC parameter is simply the object added to the List for each parameter.

Literal RPC Parameter Example:

```
RpcRequest vReq = RpcRequestFactory.getRpcRequest();
ArrayList params = new ArrayList();
params.add("I am a string");
vReq.setParams(params);
```

Reference RPC Parameter Example:

```
RpcRequest vReq = RpcRequestFactory.getRpcRequest();
ArrayList params = new ArrayList();
params.add(new RpcReferenceType("DTIME"));
vReq.setParams(params);
```

List RPC Parameter Example:

```
RpcRequest vReq = RpcRequestFactory.getRpcRequest();
ArrayList params = new ArrayList();
ArrayList nums = new ArrayList();
nums.add("3");
nums.add("5");
nums.add("4");
nums.add("1");
nums.add("7");
nums.add("8");
nums.add("2");
nums.add("6");
nums.add("9");
params.add(nums);
vReq.setParams(params);
```

Combination RPC Parameter Example:

```
RpcRequest vReq = RpcRequestFactory.getRpcRequest();
ArrayList params = new ArrayList();
ArrayList nums = new ArrayList();
nums.add("3");
nums.add("5");
nums.add("4");
nums.add("1");
nums.add("7");
nums.add("8");
nums.add("2");
nums.add("6");
nums.add("9");
params.add(nums);
params.add("I am a string");
params.add(new RpcReferenceType("DTIME"));
vReq.setParams(params);
```

8.2.4 Other Useful RpcRequest Methods

Clear Previous Request Parameters

If you are re-using a request object for additional requests, the RpcRequest.clearParams method is provided so that you can clear any existing parameters; you should call this method before attempting to set RPC parameters for subsequent requests.

```
//clear the params
vReq.clearParams();
```

Set the Message Format (Proprietary or XML)

VistALink's XML message format is based on the XML standard; its non-XML proprietary format is faster for large amounts of data. You can choose either; RpcRequest defaults to proprietary format.

```
//Set the request to use the propietary message format
vReq.setUseProprietaryMessageFormat(true);

//Set the request to use the XML message format
vReq.setUseProprietaryMessageFormat(false);
```

Set the RPC Context

If you are re-using a request object for additional requests, you can change the RPC Context with this method.

```
private static final String RPCCONTEXT = "XOBV VISTALINK TESTER";
vReq.setRpcContext(RPCCONTEXT);
```

Set the RPC Name

If you are re-using a request object for additional requests, you can change the RPC Name with this method.

```
vReq.setRpcName("XOBV TEST NOT IN CONTEXT");
```

Set the RPC Client Timeout

The RPC client timeout currently defaults to 600 seconds. You can change this value by calling the setRpcClientTimeOut method:

```
private static final int TIMEOUT = 300;
vReq.setRpcClientTimeOut(TIMEOUT);
```

For a complete list of other available RpcRequest methods, please see the javadoc for RpcRequest. Also, refer to Chapter 8 of the *VistALink Technical Manual and Package Security Guide.* Look at the subsections titled "RPC timeout handling call tags" and "RPC Time Out Process."

8.3 Response Processing

Once you have set up RpcRequest, you can execute an RPC interaction using the RpcRequest object, on the VistALinkConnection object. Doing this returns an RpcResponse object. RpcResponse is a value object that provides information about the response returned from M.

8.3.1 RpcResponseFactory Class

The RpcResponseFactory is responsible for constructing a response of type RpcResponse and handling faults that would be returned from M while processing RPC requests. This class does not need to be accessed directly, rather invoking VistaLinkConnection.executeRPC(request) will indirectly use this response factory to construct a RpcResponse.

8.3.2 RpcResponse Class

The RpcResponse class is a value object that provides information about the response returned from M. The RpcResponse object exposes methods to retrieve the results, results type and an *org.w3c.dom.*document object that contains the results if the results are in XML format.

8.3.3 Sample Code

```
J2SE Example:

//request  and response objects
RpcRequest vReq = null;
RpcResponse vResp = null;

//The Rpc Context
String rpcContext = "XOBV VISTALINK TESTER";

//The Rpc to call
String rpcName = "XOBV TEST STRING";

//Construct the request object
try{
  vReq = RpcRequestFactory.getRpcRequest(rpcContext, rpcName);
}catch(FoundationsException e){
  // process exception as needed
}
```

```
//clear the params
vReq.clearParams();

//Set the params
vReq.getParams(). setParam(1, "string", "This is a test string!");

//Set the request to use the proprietary message format
vReq.setUseProprietaryMessageFormat(true);


//Execute the Rpc and construct the response with the
//RpcResponseFactory
try{
  vResp = vistaLinkConnection.executeRPC(vReq);
}catch(VistaLinkFaultException e){
  // process exception as needed
}catch(FoundationsException e){
  // process exception as needed
}

//Display  the response
System.out.println(vResp.getResults());
```

8.3.4 Parsing RPC Results

All results from RPCs are returned as a single string, from RpcResponse.getResults().

Note: The RPC Broker defines five return types for RPCs (at the time of the current VistaLink release). The mapping between these return types and the single string returned through VistaLink are as follows:

RPC Return type	VistALink Result String Format
Single Value	as-is
Global Instance	as-is
Array	all array nodes concatenated sequentially, with each delimited by linefeed (ASCII 10) character
Global Array	all array nodes concatenated sequentially, with each delimited by linefeed (ASCII 10) character
Word Processing	each word processing "line" concatenated sequentially, separated by a linefeed (ASCII 10) character

One easy way to parse array-type results concatenated with linefeeds is with the java string tokenizer. For example:

```
StringTokenizer st = new StringTokenizer(vResp.getResults(), "\n");
int cnt = st.countTokens();
for (int i = 0; i < cnt; i++) {
   system.out.println("Result node " + i + ": " + st.nextToken());
}
```

8.3.5 XML Responses

Some newer RPCs may return their results as an XML document. The RpcResponse class provides two helper methods to turn the normal results string into an XML document. If you expect the results from an RPC to be an XML document you can call RpcResponse.isXmlResponse to confirm if the response is in XML format, and RpcResponse.getResultsDocument to convert the result string into an XML document:

```
//Get a org.w3c.dom.document object that contains the results if set
if (vReq.isXmlResponse()) {
   org.w3c.dom.document xmlDoc = null;
   try{
         xmlDoc = vResp.getResultsDocument());
   }catch(RpcResponseTypeIsNotXmlException e){
         // process exception as needed
   }catch(FoundationsException e){
         // process exception as needed
   }
}
```

Chapter 9: Utilities

9.1 VistaKernelHash

The VistaKernelHash utility class implements two static methods, encrypt and decrypt, providing the encoding / obfuscation algorithms used by the RPC Broker and Kernel to encode and decode data strings. Using these algorithms makes it harder to sniff the contents of text sent over the network. This is not, however, encryption-class encoding, nor does it protect against replay attacks of un-decoded strings, and therefore use of this algorithm should not be considered an implication or achievement of any particular security level.

For example (encoding):

```
String encodedString =
  VistaKernelHash.encrypt("some text to encode", true);
```

The second parameter is useful if the text is to be passed in a CDATA section of an XML message. If this parameter is set to true, the returned encoded strings will contain neither "]]>" nor "<![CDATA[". Otherwise, it is possible a returned encoding may contain those character sequences. If, in a reasonable number of tries, an encoded string cannot be created that doesn't contain these CDATA boundaries, an exception of type VistaKernelHashCountLimitExceededException is thrown.

For example (decoding):

```
String decodedString =
  VistaKernelHash.decrypt(encodedString);
```

9.2 XmlUtilities

9.2.1 **XmlUtilities Class**

This class contains a number of static utility methods to help developers work with XML documents, nodes, attributes and strings. These utilities are XML parser independent.

Table 6 and **Table 7**, below, briefly list the VistALink utility methods and variables, along with their description(s).

Static Method Signature	Description
`String convertXmlToStr(Document doc)`	Converts a DOM document to a string
`Document getDocumentForXmlString(String xml)`	Returns an XML DOM Document for the specified String
`Document getDocumentForXmlInputStream(InputStream xml)`	Returns an XML DOM Document for the specified InputStream
`Attr getAttr(Node node, String attrName)`	Returns the Attribute with the given attrName at node
`Node getNode(String xpathStr, Node node)`	Returns the first node at the specified XPath location

Table 6: Methods Description

Static Final Variables	Description
`String XML_HEADER`	Represents the default header used for all xml documents that communicate with an M server via VistALink. It is important to use this header as keeps the client and M server in sync.

Table 7: Final Variables Description

9.2.2 Sample Code

```
J2SE Example:

String xmlStr = XmlUtilities.XML_HEADER
      + "<VistaLink messageType='"
      + RpcRequest.GOV_VA_MED_RPC_REQUEST
      + "'"
      + " mode='singleton'"
      + " version='1.0'"
      + " xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'"
      + " xsi:noNamespaceSchemaLocation='rpcRequest.xsd'"
      + " xmlns='http://med.va.gov/Foundations'"
      +">"
      + "  <RpcHandler version='1.0'/>"
      + "  <Request rpcName='' rpcClientTimeOut='600' version='1.0' >"
      + "    <RpcContext></RpcContext>"
      + "    <Params><Param type=’array’ ></Param></Params>"
      + "  </Request>"
      + "</VistaLink>";

Document doc = XmlUtilities.getDocumentForXmlString(xmlStr);

Node param = XmlUtilities.getNode("/VistaLink/Request/Params/Param",
requestDoc);

String type = XmlUtilities.getAttr(param, "type").getValue();
String xmlCopy = XmlUtilities.convertXmlToStr(xmlDoc);
...
FileInputStream xmlStream = FileInputStream(“myRequest.xml”);
Document myReq = XmlUtilities.getDocumentForXmlInputStream(xmlStream);
...
```

9.3 AuditTimer

gov.va.med.foundations.utilities.AuditTimer is used within VistALink to capture and log information on how long it took to execute a specific VistALink interaction using lo4j-logging capabilities.

Special logger *gov.va.med.foundations.adapter.spi.VistaSocketConnection.AuditLog* is used to output this information. Applications can use *AuditTimer* independently if they want to report timer information for various processing requests.

Two type of constructors can be used to construct *AuditTimer* instance:
public AuditTimer() – default logger "*gov.va.med.foundations.utilities.AuditTimer*" will be used.
public AuditTimer(Logger logger) – application specific *logger* will be used.

AuditTimer logs milliseconds elapsed between *start()* and *stop()* calls.

Number of elapsed milliseconds can be retrieved using *getTimeElapsedMillis()*.

Logging can be done using either *log()* method:
public void log()
public void log(String message) – Since *logger* can be passed into the constructor and output pattern for a specific logger can be configured using log4j configuration file, there should be no need to pass info message. Instead different loggers should be used.

Sample code:

```
import gov.va.med.foundations.utilities.AuditTimer;

public class AuditTimerTest {
	private static AuditTimer timer = null;
	private static Logger auditLogger =
Logger.getLogger(AuditTimerTest.class.getName() + ".AuditLog");

	public static void main(String[] args) {
		// Initialize Log4j configuration
		DOMConfigurator.configureAndWatch("props/log4jConfig.xml",
10000);

		timer = new AuditTimer(auditLogger);
		// Start timer
		timer.start();

		// ... perform your operations

		// Stop timer
		timer.stop();

		// Log elapsed time information
		timer.log();

		// Get time elapsed if not for logging purposes
		long timeElapsed = timer.getTimeElapsedMillis();
	}
}
```

The following is a snippet from the log4j configuration file:

```
  <appender name="auditTimerTestConsoleAppender"
class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="Audit time - %m%n"/>
    </layout>
  </appender>

  <logger
name="gov.va.med.foundations.utilities.test.AuditTimerTest.AuditLog" >
        <level value="debug" />
      <appender-ref ref="auditTimerTestConsoleAppender"/>
  </logger>
```

Chapter 10: Exceptions

VistALink, like any other Java application, uses exceptions to indicate various error conditions that could occur during excecution.

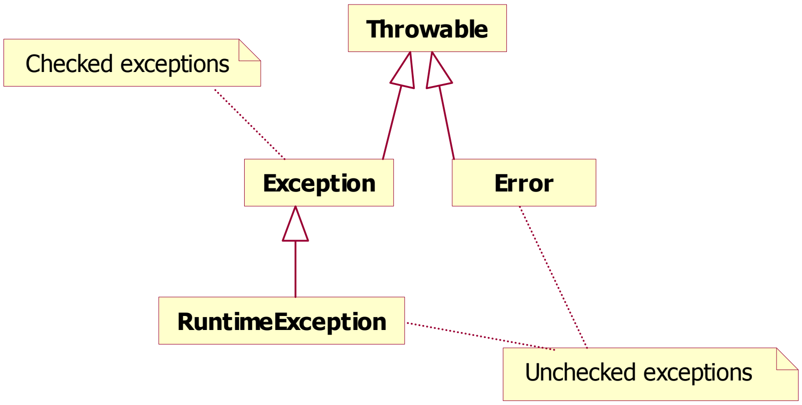
10.1 Checked and Unchecked Exceptions

There are two types of exceptions in Java programming language – *checked* and un*checked* exceptions.

Checked exceptions have to be declared in the method signature *throws* clause if they are thrown by the method. *Checked* exceptions have to be explicitly caught by the caller within a *try / catch* block.

Unchecked exceptions do not have to be declared in the method signature if they are thrown by the method. *Unchecked* exceptions do not need to be explicitly cought by the caller. *Unchecked* exceptions can be caught by the caller even if method do not explicitly throw them.

The diagram below depicts a class hierarchy of base Java exception classes:



java.lang.Error – unchecked - reserved to JVM exceptions - memory, out of stack etc.

java.lang.RuntimeException – unchecked - reserved for JVM exceptions that are not as fatal as *java.lang.Error* exceptions, such as array index out of bound, null pointer exception etc.

class java.lang.Exception – checked - application level exceptions

VistALink throws only *checked* exceptions.

10.2 Catching Exceptions

It is important to remember that an application calling a method that (in turn) throws exception *A* can do two things:

1. catch this exception *A*
2. catch any exception *AB* that is subclassed from exception *A*, even though calling a method declaration only declares to throw a parent exception

Example:

```
public class ParentException extends Exception {
}
public class SubException extends ParentException {
}
public class ExceptionSample {

      public static void test() throws ParentException {
         throw new SubException();
      }

      public static void main(String[] args) {
            try {
                  ExceptionSample.test();
            } catch (SubException e) {
                  System.out.println("Caught SubException exception");
            } catch (ParentException e) {
                  System.out.println("Caught ParentException exception");
            }
      }
}
```

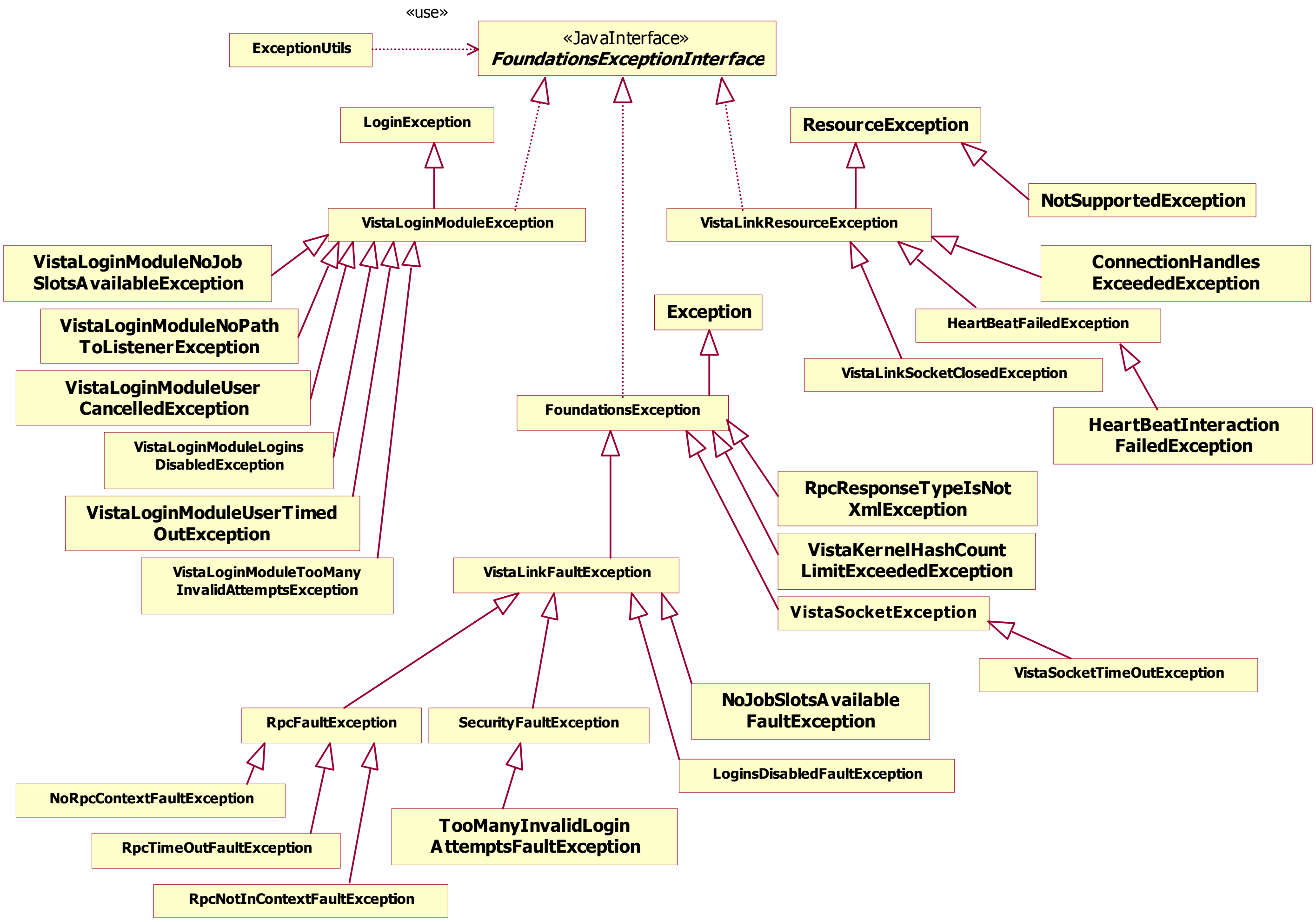
In this example, we are declaring *ParentException* and its subclass *SubException*. *ExceptionSample* class has a method *test()* that declares to throw *ParentException*. Method *test()* implementation instead throws a more specific exception *SubException*. The *test()* method could choose to declare the fact that it is throwing both *ParentException* and *SubException*, but that is not required by the Java specifications.

The *test()* method caller can only catch *ParentException*. This takes care of catching *ParentException* and all *ParentException* subclasses. But if *test()* method caller knows that *test()* method throws a more specific exception, (a subclass of the *ParentException*) then the caller can choose to catch a more specific *SubException*, even though *test()* method does not explicitly declare the fact that it throws *SubException*.

This is an important point, as both the VistALink security modules and the VistALink connector modules often throw more specific VistALink exceptions, even though those exceptions are not declared to be thrown and only parent exceptions are declared to be thrown from VistALink methods.

10.3 VistALink Exception Hierarchy

Below is a VistALink exception class diagram.


«use»
ExceptionUtils
«JavaInterface»
FoundationsExceptionInterface
LoginException
ResourceException
NotSupportedException
VistaLoginModuleException
VistaLinkResourceException
VistaLoginModuleNoJobSlotsAvailableException
VistaLoginModuleNoPathToListenerException
VistaLoginModuleUserCancelledException
VistaLoginModuleLoginsDisabledException
VistaLoginModuleUserTimedOutException
VistaLoginModuleTooManyInvalidAttemptsException
ConnectionHandlesExceededException
Exception
HeartBeatFailedException
VistaLinkSocketClosedException
HeartBeatInteractionFailedException
FoundationsException
RpcResponseTypeIsNotXmlException
VistaKernelHashCountLimitExceededException
VistaSocketException
VistaSocketTimeOutException
VistaLinkFaultException
NoJobSlotsAvailableFaultException
RpcFaultException
SecurityFaultException
LoginsDisabledFaultException
NoRpcContextFaultException
RpcTimeOutFaultException
TooManyInvalidLoginAttemptsFaultException
RpcNotInContextFaultException

Since VistALink implements various Java specificitions such as JAAS and J2EE Connectors, each Java specification dictates usage of a specific base exception class. To be able to work with all types of exceptions we are defining one unifying exception interface *gov.va.med.foundations.utilities.FoundationsExceptionInterface*:

Two methods are defined in this interface: *getFullStackTrace()* and *getNestedException()*. These methods are used by utility classes such as *gov.va.med.foundations.utilities.ExceptionUtils* to retrieve nested exception information.

10.4 JAAS Exceptions

JAAS requires LoginModules to throw *javax.security.auth.login.LoginException* from JAAS classes implementation methods.

VistALink security classes use *gov.va.med.foundations.security.vistalink.VistaLoginModuleException* that extends *javax.security.auth.login.LoginException*.

A more specific JAAS exceptions are available for VistALink users to catch and reside in *gov.va.med.foundations.security.vistalink* package (see exception class diagram provided above):

VistaLoginModuleLoginsDisabledException

VistaLoginModuleNoJobSlotsAvailableException

VistaLoginModuleNoPathToListenerException

VistaLoginModuleTooManyInvalidAttemptsException

VistaLoginModuleUserCancelledException

VistaLoginModuleUserTimedOutException

10.5 J2EE Connectors Exceptions

J2EE Connectors requires adapter methods to throw *javax.resource.ResourceException.*

10.5.1 VistaLinkResourceException

VistALink implements *gov.va.med.foundations.adapter.cci.VistaLinkResourceException* that extends *javax.resource.ResourceException*. *VistaLinkResourceException* is thrown from all J2EE Connectors required methods as well as any custom method in VistALink that implements connection management interfaces.

VistaLinkResourceException more specific exception subclasses include: (see exception class diagram provided above)

ConnectionHandlesExceededException

VistaLinkSocketClosedException

HeartBeatFailedException and *HeartBeatInteractionFailedException*.

See adapter section <link> of the document for more details.

10.5.2 FoundationsException

J2EE Connectors defines optional record and interaction management interfaces that are not implemented in VistALink. Instead VistALink uses *VistaLinkConnection::executeRPC()*, *VistaLinkConnection::executeInteration()* and classes in *gov.va.med.foundations.adapter.record* package to implement interaction and record management. These methods are not governed by J2EE Connectors. Hence VistALink uses *gov.va.med.foundations.utilities.FoundationsException* and it's subclasses in these methods.

10.5.3 VistaLinkFaultException

VistALink communications with M can produce exceptions that originate from the Java code side. In that case *VistaLinkResourceException* and *FoundationsException* will be thrown that are described above.

VistALink communications with M can also produce exceptions that originate from the M side. In those cases a Fault message is sent back to VistALink from M. Fault messages are parsed and *gov.va.med.foundations.adapter.record.VistaLinkFaultException* that extends *FoundationsException* are constructed to be thrown in those cases. *VistaLinkFaultException* will have all the information that is sent back from the M side to Java, including: *faultCode*, *faultString*, *faultActor*, *errorCode*, *errorType* and *errorMessage*.

VistaLinkFaultException more specific exception subclasses include: (see exception class diagram provided above)

NoJobSlotsAvailableFaultException

LoginsDisabledFaultException

SecurityFaultException that in turn have more specific subclasses.

RpcFaultException that in turn have more specific subclasses.

See adapter section <link> of the document for more details.

10.5.4 Common Exception Interface

gov.va.med.foundations.utilities.FoundationsExceptionInterface is defined to be able to have common interface for all types of VistALink exceptions. Implementation of this interface allows *gov.va.med.foundations.utilities.ExceptionUtils* to work exceptions no matter what they inherit from.

10.5.5 Exception Nesting

Exception nesting is a technique to nest exceptions that is used to collect full information about the error that occurred in processing method call.

Example: Exception *A* can be thrown from a library. Client code catches the exception *A* and rethrows new exception *B* while preserving the original exception *A* within new exception's member variables.

This new exception *B* once again could be cought by some other client code that could throw new exception *C* while preserving cought exception *B* within exception's *C* member variables:

A nested within *B* nested within *C*.

This way all exceptions that are cought and rethrown are kept as a linked exception list allowing us not to loose any information about the error origination and how it is handled by the code.

JDK 1.4 has native support for exception nesting while JDK 1.3 does not have native support for exception nesting. Since VistALink runs both on JDK 1.3 and 1.4 VistALink has implemented custom exception nesting framework.

All VistALink base exceptions classes implement exception nesting:

gov.va.med.foundations.utilities.FoundationsException

gov.va.med.foundations.adapter.cci.VistaLinkResourceException

gov.va.med.foundations.security.vistalink.VistaLoginModuleException

10.6 Working With Nested Exceptions

If your code catches one of the above exceptions here are some features that can be expected:

exception.getMessage() returns all nested exception messages in the folling format:

```
Wrapper exception;
Root cause exception:
java.lang.Exception: Here is my nested exception.
```

exception.printStackTrace() will print both nested exception messages and full stack trace:

```
gov.va.med.foundations.utilities.FoundationsException: Wrapper exception;
      Root cause exception:
      java.lang.Exception: Here is my nested exception.
java.lang.Exception: Here is my nested exception.
      at
gov.va.med.foundations.utilities.test.FoundationsExceptionTest.t2(Foundati
onsExceptionTest.java:106)
      at
gov.va.med.foundations.utilities.test.FoundationsExceptionTest.t1(Foundati
onsExceptionTest.java:109)
      at
gov.va.med.foundations.utilities.test.FoundationsExceptionTest.testConstru
ctorStrExc(FoundationsExceptionTest.java:128)
      at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
      at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
      at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
      at java.lang.reflect.Method.invoke(Unknown Source)
      at junit.framework.TestCase.runTest(TestCase.Java:154)
      at junit.framework.TestCase.runBare(TestCase.Java:127)
      at junit.framework.TestResult$1.protect(TestResult.Java:106)
      at junit.framework.TestResult.runProtected(TestResult.Java:124)
      at junit.framework.TestResult.run(TestResult.Java:109)
      at junit.framework.TestCase.run(TestCase.Java:118)
      at junit.framework.TestSuite.runTest(TestSuite.Java:208)
      at junit.framework.TestSuite.run(TestSuite.Java:203)
      at junit.swingui.TestRunner$16.run(TestRunner.Java:623)
```

exception.getNestedException() will return nested exception. Note: do not use this method to unwind nested exception linked list as there is helper method in *ExceptionUtils.getNestedExceptionByClass()* that does just that.

See *ExceptionUtils* below for more methods that can be used working with exceptions.

10.6.1 ExceptionUtils

gov.va.med.foundations.utilities.ExceptionUtils is a utility class that contains static helper methods such as *getFullStackTrace()* and *getNestedExceptionByClass()* to help unwind nested exception stack trace.

10.6.2 ExceptionUtils:: getFullStackTrace(Throwable e)

ExceptionUtils:: getFullStackTrace(Throwable e) returns full stack trace in case you need to pring stack trace for exception *e* to some other place other than system output console such as HTML error page or Swing text box. To print stack trace to system output console just use *exception.printStackTrace()*.

Notice we are using *Throwable* instead of *Exception* here as this will allow us to print stack trace even for *Errors*.

getFullStackTrace method prints out both, nested exception message set as well as full stack trace. Which means, if you are using this method to return full stack trace, error message will be part of it and you don't need to use *excetion.getMessage()*.

10.6.3 ExceptionUtils:: getNestedExceptionByClass()

ExceptionUtils:: getNestedExceptionByClass(Throwable e, Class exceptionClass) returns an exception if an exception of a class type *exceptionClass* exists in the exception *e* nested exception linked list.

So if your nested exceptions look like this:

(a instance of A) nested within (*b instance of B)* nested within (*c instance of C)*.

Then a calls:

ExceptionUtils:: getNestedExceptionByClass(c, C) returns *c*.

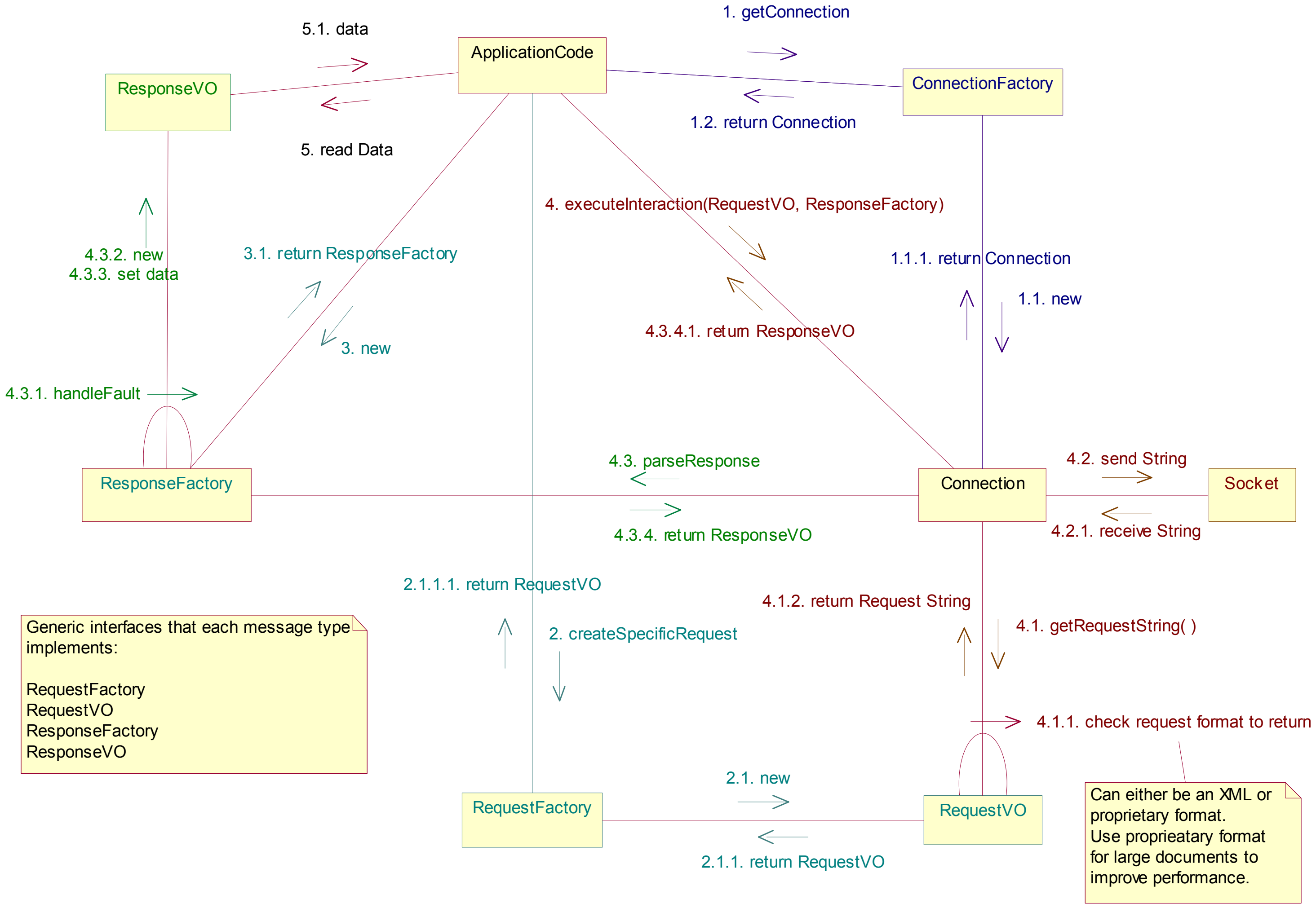
ExceptionUtils:: getNestedExceptionByClass(c, B) returns *b*.

ExceptionUtils:: getNestedExceptionByClass(c, A) returns *a*.

Appendix – Java and M-Side Request / Response Processing

Java-Side VistALink Request / Response Processing

VistALink implements pluggable Request/Response processing framework on the Java side. See the collaboration diagram below for an overview.



Generic interfaces that each message type implements:

RequestFactory
RequestVO
ResponseFactory
ResponseVO

Can either be an XML or proprietary format.
Use propriеatary format for large documents to improve performance.

Message Types Supported By VistALink

1. System messages that are reserved for VistALink internal communications such as for heartbeat interactions.
2. Security messages that are reserved for VistALink security module.
3. RPC messages that are designed to handle MUMPS RPC execution.

Applications using VistALink rely on RPC communication to produce RPC messages. Click http://www.va.gov/vdl/ and see the *RPC Broker – Getting Started With the Broker Development Kit (BDK)* document for details. It is possible for applications to plug-in their own extensions to RPC Response processing by sub-classing the RpcResponseFactory and RpcResponse classes. This is not recommended, as it introduces tight dependencies between VistALink implementation and custom extensions. Instead of implementing of *reuse by inheritance*, it is advisable to implement *reuse by encapsulation*. Using this approach, applications use *containment* for extending VistALink functionality. Obviously, the VistALink team will maintain the practice of implementing *reuse by inheritance* for future VistALink extensions.

M-Side VistALink Request / Response Processing

VistALink provides TCP/IP transport functionality that enable synchronous communication 1) from non-M based systems to M based VistA servers and 2) from M VistA servers to non-M systems (typically Java/J2EE based systems).

The communications are in the form of requests.

For requests received by the M based system, VistALink provides 1) a TCP/IP listener to receive the raw request bytes, 2) a Request Manager to parse, manage and direct these requests to handlers and 3) two native request handlers, one handles system-type requests, like client heartbeats and close socket requests and the other handler processes RPC requests.

Other packages can also supply handlers. For example, the VistALink Security supplies a handler used to process VistA security related requests. (Note: VistALink Security is distributed as part of the XOB KIDS file.)

VistALink allows request formats to be in (a) XML that conforms to an XML Schema, (b) a proprietary format (package must supply proprietary parser) or (c) both a and b. **Table 8**, below, details these possible formats.

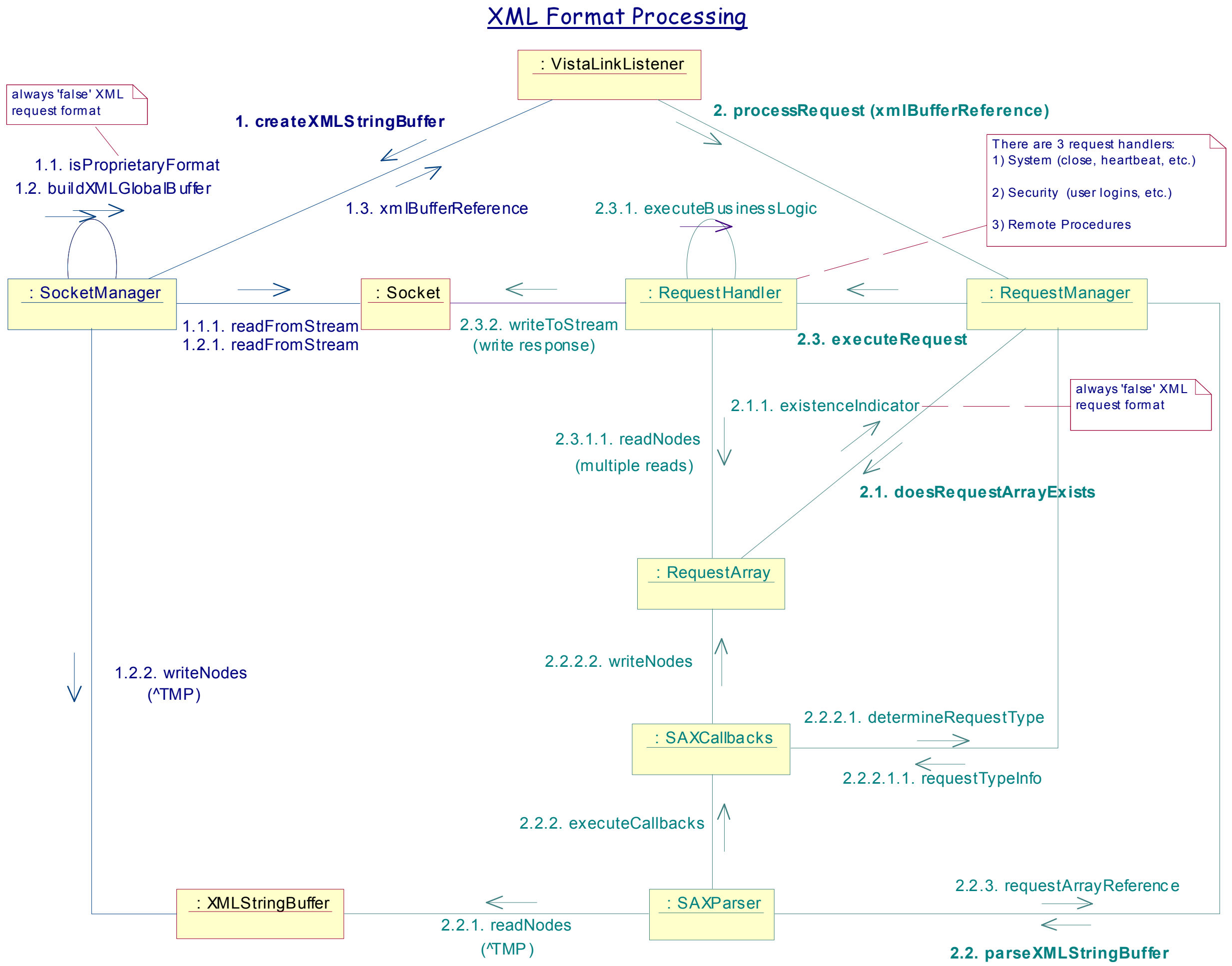
Request Handler	Package	XML Requests	Proprietary Requests
System	VistALink (XOBV)	Yes	No
RPC	VistALink (XOBV)	Yes	Yes
Security	VistALink Security (XOBS)	Yes	No

Table 8: Request Handler Summary

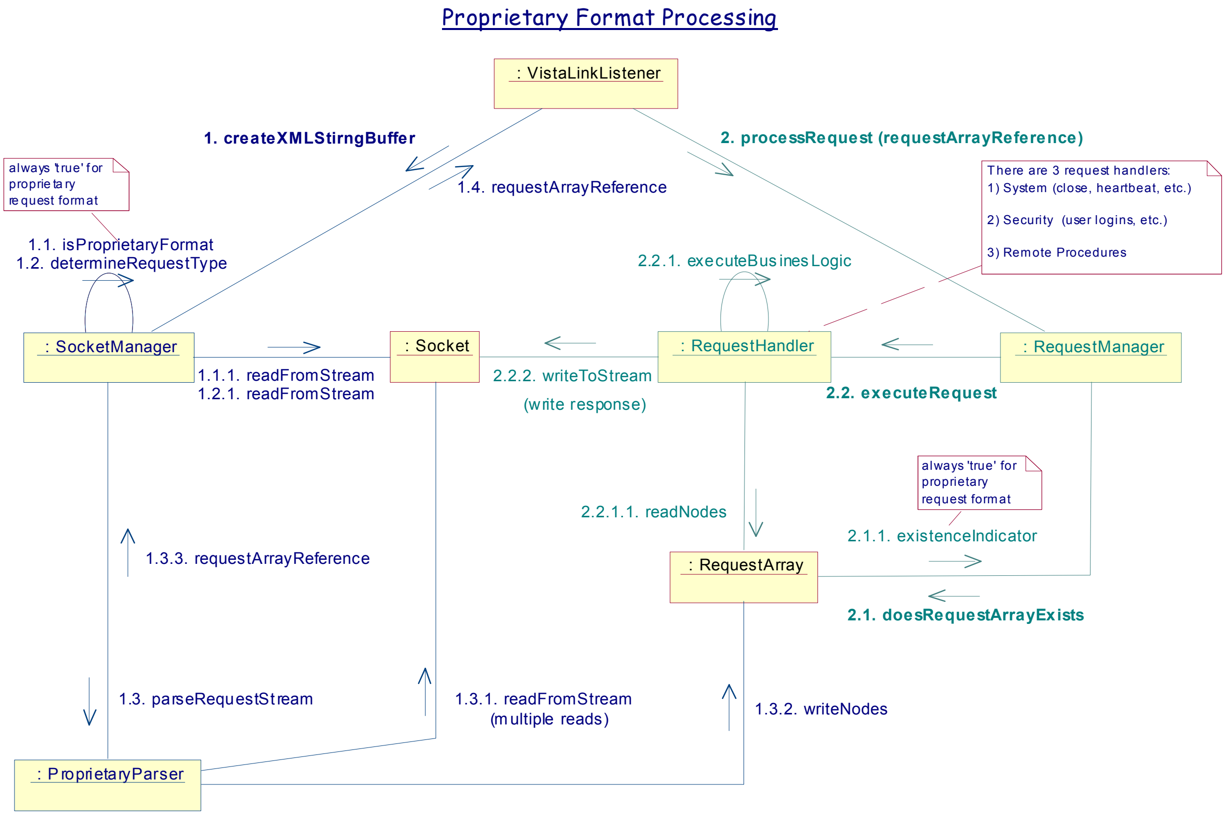
Note: As of April 2003, VistALink supports Java-to-M communication ONLY.

The diagrams on the next two pages of this manual will help the user understand how VistALink processes requests received by an M server.

This diagram illustrates the processing of requests in XML format.



This diagram illustrates the processing of requests in a proprietary format.


Proprietary Format Processing
: VistaLinkListener
1. createXMLStirngBuffer
2. processRequest (requestArrayReference)
always 'true' for proprietary request format
1.4. requestArrayReference
There are 3 request handlers:
1) System (close, heartbeat, etc.)
2) Security (user logins, etc.)
3) Remote Procedures
1.1. isProprietaryFormat
1.2. determineRequestType
2.2.1. executeBusinesLogic
: SocketManager
: Socket
: RequestHandler
: RequestManager
1.1.1. readFromStream
1.2.1. readFromStream
2.2.2. writeToStream
(write response)
2.2. executeRequest
always 'true' for proprietary request format
2.2.1.1. readNodes
2.1.1. existenceIndicator
1.3.3. requestArrayReference
: RequestArray
2.1. doesRequestArrayExists
1.3. parseRequestStream
1.3.1. readFromStream
(multiple reads)
1.3.2. writeNodes
: ProprietaryParser

Index