HealtheVet-VistA

Electronic Signature (ESig)
Version 1.0

Developer Guide

November 2006

U.S. Department of Veterans Affairs
Health Systems Design & Development

Revision History

Date	Revision	Description	Contacts
November 2006	1.0	Release	**Project Manager and Analyst:** Dawn Clark
			Developer: Michael Ogi
			Technical Writer: Jim Alexander

Table of Contents

List of Figures

List of Tables

Contents

ESig 1.0 Developer Guide

1 Introduction

1.1 ESig Overview

As Health*e*Vet-VistA developers migrate VistA applications to modern technologies, interim solutions may be required until enterprise solutions are mature and stable. The Electronic Signature (ESig) service provides an interim solution for the use of electronic codes in place of wet signatures while Health*e*Vet-VistA's security infrastructure and architecture are being defined. The service duplicates for Java applications (J2EE or J2SE) the Kernel 8.0 electronic signature functionality currently used by VistA/M applications.

ESig furnishes a standard, consistent set of APIs that Health*e*Vet-VistA developers can implement to provide users access to electronic signature data stored on VistA/M systems. ESig APIs make calls from Java applications to VistA/M systems to retrieve, validate, and store electronic signature codes and signature block information (name, title, office phone, etc.). Additional Java APIs provide encoding/decoding, hash, and checksum calculation utilities, but do not interact with the VistA/M system.

Applications that implement the ESig service must provide a user interface (UI) to prompt users for their secret codes when authorizing orders, prescriptions, financial transactions, or other business processes. Users may also need the UI to create or modify their code or signature block data.

1.1.1 VistALink 1.5 Dependency

ESig requires the VistALink 1.5 service, which provides the transport layer enabling communication between a Java application and a VistA/M system.

The figure below shows ESig APIs communicating with VistA through VistALink 1.5. When a Health*e*Vet user signs on successfully, the connection from the application to VistA via VistALink is established. Consuming applications pass the VistaLinkConnection object to the ESig APIs that communicate with the VistA server.
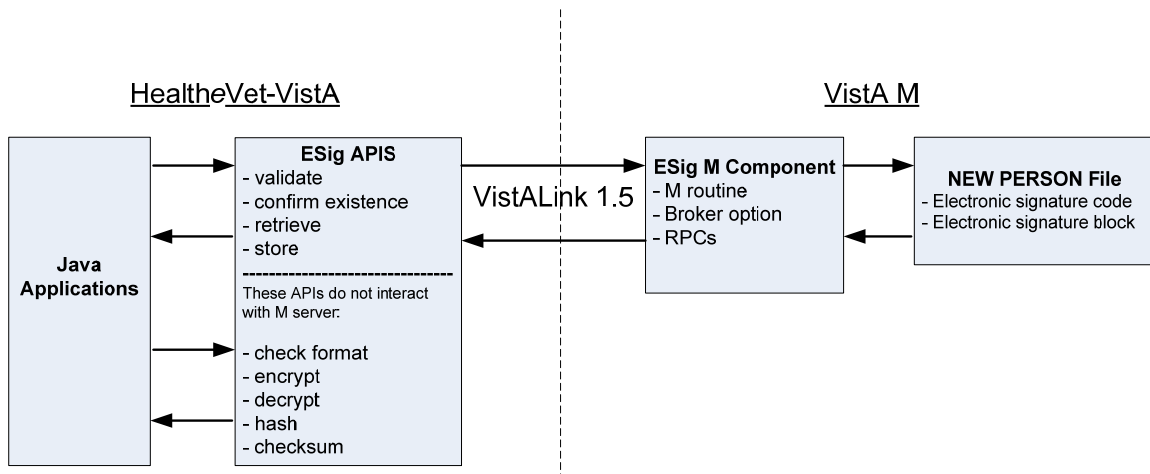
Figure 1-1. ESig Architecture

1.1.2 Installation

Health*e*Vet ESig consists of three parts:

- An M package containing a routine, a Broker option, and a set of Remote Procedures for accessing electronic signature codes and related data in the Kernel's NEW PERSON (#200) file

- A JAR file containing a set of Java APIs for passing and receiving electronic signature related information from M, and for performing hashing, encryption, and decryption of strings. For ESig functionality to work, the ESig JAR file must be present on an application's classpath.

- Sample Java Swing, client console, and JSP utility applications to test or verify installation and configuration of the ESig components. These are included in the ESig distribution.

Although ESig is a Health*e*Vet-VistA application, the only installation required is the KIDS build on the VistA/M server. Health*e*Vet-VistA applications requiring electronic signature functionality will include the ESig JAR file in their classpath. The JAR file contains APIs to perform ESig functions, including calling the VistA/M database.

Application developers and testers may want to deploy the sample ESig applications to client workstations (J2SE) or application servers (J2EE) to test the installation of the M server pieces. Instructions for deploying the sample applications are included in this guide.

1.1.3 ESig Technical Summary

The table below summarizes technical information about ESig in Development and Production environments.

Table 1-1. ESIG Technical Information

Overview	Technologies Used	Dependencies	Development Tools
• Provides Health*e*Vet applications access to Kernel electronic signature APIs • Supports J2EE and J2SE implementations • Requires ESIG KIDS build installation on VistA/M server • Health*e*Vet application provides any necessary user interfaces • Distributed with feature-complete sample applications (J2SE and J2EE) • Sample J2EE application can be deployed to admin/managed servers/clusters	• Caché 5.x (NT and VMS) • BEA WebLogic Server 8.1.4 (on Windows and Linux) • J2EE 1.3 • J2SE SDK 1.4 • Log4j • Windows 2000/XP • Red Hat Linux • XML	• VistALink 1.5 • Kernel 8.0 • Kernel Toolkit 7.3 • VA FileMan 22.0 • RPC Broker 1.1	• Apache Ant • Eclipse IDE with MyEclipse IDE Plug-in • IBM Rational ClearQuest • IBM Rational Rose/XDE • IBM Rational Unified Process • JUnit • Microsoft Visual SourceSafe • XDoclet

1.2 Using this Guide

1.2.1 Purpose/Audience

This document provides detailed information about ESig APIs and exceptions for developers intending to use ESig functionality in their applications. It also contains instructions for deploying sample J2EE (application server) and J2SE (client-server) applications. These sample applications can be used by developers and testers to exercise ESig APIs from the host application.

1.2.2 Text Conventions

The table below summarizes specialized use of typographical styles in this document.

Table 1-2. Text Conventions

Convention	Explanation	Example
ALL CAPS	M file, routine, variable, field, menu, field, and security key names.	Developers should be assigned the XUPROGMODE security key. The option [XOBE ESIG USER] may be added to the menu.
Boldface	Java file and directory names, particularly the first time they are mentioned in a passage.	Locate the **javadoc** folder and open your browser to the **index.html** file.
	Java GUI buttons.	Press **Enter**.
	Used in M dialog examples to show user entries.	Enter a Host File: **XOBE_1_.KID**
Courier font	Java class, method, or variable names	ESigConnectionException
<Angle brackets>	M key entries.	<Enter>
	In Java-related text, indicates information that is unknown or must be supplied by the user.	Locate the **jaas.config** file in the **<ESIG_SAMPLE_APP>** folder.
"Quotation marks"	Verbatim user entries in Java-related instructions.	You should name the file "log4j.xml".

The following symbols appear throughout the documentation to alert the reader to special information or conditions.

Symbol	Description
	Used to inform the reader of general information and reference material.
	Used to caution the reader to take special notice of critical information

1.3 Additional Resources

1.3.1 ESig Reference Materials

The following documents are included in the ESig documentation set:

- *ESig 1.0 Installation Guide* – Prerequisites and instructions for installing the ESig KIDS build on a VistA/M server.

- *ESig 1.0 Developer Guide* – Detailed information about ESig APIs and exceptions, for developers intending to use ESig APIs in their applications. This document includes instructions useful to developers, quality assurance, and testers

for deploying sample J2EE (application server) and J2SE (client-server) applications. These sample applications test the ESig APIs used by the host application.

- *ESig 1.0 System Management Guide* – VistA/M-side system and security information.

Because ESig APIs communicate with VistA/M systems through VistALink and Kernel RPCs, the following documentation is highly recommended:

- VistALink 1.5 documentation: *Developer Guide*, *Installation Guide*, and *System Management Guide*.

- RPC documentation: *RPC Broker Getting Started with the Broker Development Kit (BDK,) RPC Broker Developer's Guide* (i.e., BROKER.HLP, online help designed for programmers, distributed in the BDK)

- *Kernel v.8.0 Systems Manual*

ESig, VistALink, and RPC Broker documents are available on any of the Office of Information FTP directories as well as the VHA Document Library (VDL) at http://www.va.gov/vdl/.

1.3.2 Online Technical Information

1.3.2.1 <u>VistA/M Help</u>

After the ESig KIDS build is installed on the VistA/M server, developers and system administrators can use the standard Kernel/FileMan utilities for printouts of the installed components.

VistA software has online help and commonly used system default prompts. In roll-and-scroll mode, users are strongly encouraged to enter question marks at any response prompt. At the end of the help display, you are immediately returned to the point from which you started. This is an easy way to learn about any aspect of VistA software.

To retrieve online documentation in the form of Help in VistA roll-and-scroll software:

- Enter a single question mark ("**?**") at a field/prompt to obtain a brief description. If a field is a pointer, entering one question mark ("**?**") displays the HELP PROMPT field contents and a list of choices, if the list is short. If the list is long, the user is asked if the entire list should be displayed.

   A YES response invokes the display. The display can be given a starting point by prefacing the starting point with an up-arrow ("**^**") as a response. For example, **^M** would start an alphabetic listing at the letter M instead of the letter A, while **^127** would start any listing at the 127th entry.

- Enter two question marks ("**??**") at a field/prompt for a more detailed description. Also, if a field is a pointer, entering two question marks displays the HELP PROMPT field contents and the list of choices.

- Enter three question marks ("**???**") at a field/prompt to invoke any additional Help text that may be stored in Help Frames.

1.3.2.2 VistA/M Data Dictionary Listings

Technical information about files and the fields in files is stored in data dictionaries. To print formatted data dictionaries, go to the VA FileMan v.22.0 Advanced User Documentation (http://vista.med.va.gov/fileman/docs/u2/index.shtml) and click on the List File Attributes link on the left frame.

1.3.2.3 Javadocs

Java class and package documentation is included in the ESig distribution file. Locate the **javadoc** folder and open your browser to the **index.html** file.

> To learn more about Javadoc files, refer to Sun's Javadoc Tool Home Page at: http://java.sun.com/j2se/javadoc/.

> **DISCLAIMER:** The appearance of external hyperlink references in this manual does not constitute endorsement by the Department of Veterans Affairs of the information, products, or services on the Website. The VHA does not exercise any editorial control over the information you may find at these locations.

2 ESig APIs

2.1 ESig Java Classes

ESig 1.0 includes the Java classes described in the sections below. (For more information about ESig Java packages and classes, see the Javadocs contained in the **javadoc** folder of the ESig distribution.)

2.1.1 esig-1.0.0.nnn.jar

2.1.1.1 Package: gov.va.med. esig.utilities

This package contains the following classes for validating, retrieving, and setting electronic signature codes and related data:

- **ESigDataAccess** – This class contains the static methods that access electronic signature data on the server.

- **ESigEncryption** – This class implements the static methods that provide the encoding and decoding algorithms for electronic signatures.

- **ESigValidation** – This class implements the static methods for validating a user-supplied electronic signature code.

The package also contains the following exceptions:

- **ESigConnectionException** – This exception class is thrown when an error occurs while attempting to connect to the server that contains the electronic signature data.

- **ESigException** – This base exception class implements exception nesting.

- **ESigInvalidFormatException** – This exception class is thrown if an attempt is made to save an electronic signature code on the server when its format invalid.

- **ESigNoElectronicSignatureDefinedException** – This exception class is thrown if an attempt is made to validate a user-supplied electronic signature code when the user has no electronic signature code currently defined on the server.

- **ESigNotAValidUserException** – This exception class is thrown if an attempt is made to access the electronic signature data on the server when the user is not defined on the server.

- **ESigUnchangedElectronicSignatureException** – This exception class is thrown if an attempt is made to update the electronic signature code for the user on the server when the new electronic signature is the same as the old one.

2.1.2 esigSamples-1.0.0.nnn.jar

2.1.2.1 <u>Package: gov.va.med. esig.samples</u>

This package contains the ESig sample programs, which exercise the ESig toolset for validating, retrieving, and setting electronic signature codes and related data. The sample programs are composed of the following classes:

- **DialogConfirm** – This class is a Swing Dialog to display information to the user.

- **ESigApiSwingTester** – This is a Swing-based developer example application that demonstrates the ESig toolset functionality.

- **ESigSample** – This is a console application that runs in a DOS window and demonstrates the ESig toolset functionality.

2.2   ESig Java APIs – Details

The package **gov.va.med.esig.utilities** contains the three classes in the following table.

Table 2-1. ESig API Classes

ESigDataAccess	Implements static methods that access electronic signature code data on the M server
ESigEncryption	Implements static methods to provide the encoding and decoding algorithms similar to those provided by Kernel v. 8.0.
ESigValidation	Implements static methods that validate a user-supplied electronic signature code.

The APIs in these classes are also documented in the Javadoc documentation included with the Electronic Signature distribution. To view Javadoc, open the file **<DISTRIBUTION_HOME>/javadoc/index.html** in your web browser.

2.2.1 Class ESigDataAccess

The ESigDataAccess class contains static methods that access electronic signature codes and related data on the VistA M server.

isDefined

The isDefined method returns true if the user has an electronic signature code defined on the M server.

```
public static final boolean isDefined
    (gov.va.med.vistalink.adapter.cci.VistaLinkConnection connection)
        throws ESigConnectionException,
            ESigNotAValidUserException
```

Parameters:
>   connection - The `VistaLinkConnection` handle.

Returns:
>   `true` if the user has an Electronic Signature Code defined on the M server.

Throws:
>   `ESigConnectionException` - if the RPC request fails.
>   `ESigNotAValidUserException` - if the DUZ of the user does not correspond to a valid entry in the New Person file.

Example:

```
try {
    if (ESigDataAccess.isDefined(myConnection)) {
        System.out.println("Your electronic signature code is defined
            on the M server.");
    } else {
        System.out.println("You currently have no electronic signature
            code defined.");
    }
} catch (FoundationsException e) {
    System.out.println(e.getMessage());
}
```

getESigCode

The `getESigCode` method retrieves the encrypted electronic signature code from the M server.

```
public static final java.lang.String getESigCode
    (gov.va.med.vistalink.adapter.cci.VistaLinkConnection connection)
        throws ESigConnectionException,
            ESigNotAValidUserException
```

Parameters:
>   connection - The `VistaLinkConnection` handle.

Returns:
>   A String that contains the user's encrypted Electronic Signature Code.

Throws:
>   `ESigConnectionException` - if the RPC request fails.
>   `ESigNotAValidUserException` - if the DUZ of the user does not correspond to a valid entry in the New Person file.

Example:

```
try {
    String eSig = ESigDataAccess.getESigCode(myConnection);
    System.out.println("  ESig obtained from VistA: " + eSig);
} catch (Exception e) {
    System.out.println(e.getMessage());
```

saveESigCode

The overloaded `saveESigCode` method take the unencrypted electronic signature code either in a character array or a String, and saves the encrypted form of the electronic signature code in the New Person file on the M server.

```
public static final void saveESigCode(char[] eSigCode,
    gov.va.med.vistalink.adapter.cci.VistaLinkConnection connection)
        throws ESigConnectionException,
            ESigNotAValidUserException,
            ESigUnchangedElectronicSignatureException,
            ESigInvalidFormatException

public static final void saveESigCode(java.lang.String eSigCode,
    gov.va.med.vistalink.adapter.cci.VistaLinkConnection connection)
        throws ESigConnectionException,
            ESigNotAValidUserException,
            ESigUnchangedElectronicSignatureException,
            ESigInvalidFormatException
```

Parameters:
> `eSigCode` - An array of characters or a String that contains the user-supplied (unencrypted) electronic signature code.
> `connection` - The VistaLinkConnection handle.

Throws:
> `ESigConnectionException` - if the RPC request fails.
> `ESigNotAValidUserException` - if the DUZ of the user does not correspond to a valid entry in the New Person file.
> `ESigUnchangedElectronicSignatureException` - if the electronic signature on the M server is the same as the electronic signature code passed in.
> `ESigInvalidFormatException` - if the format of the electronic signature code passed in is not valid. VistA electronic signatures codes must be between 6 and 20 characters in length, and cannot contain control characters. That is, they must contain only the printable characters in the 7-bit ASCII character set, decimal ASCII values 32 through 126

Example:

```
try {
    String esig = "NEW ESIG VALUE";
```

```
    System.out.println("Value attempting to save: " + esig);

    ESigDataAccess.saveESigCode(esig, myConnection);
    System.out.println("Value " + esig + " saved successfully.");

} catch (Exception e) {
    System.out.println(e.getMessage());
}
```

getESigData

The `getESigData` method obtains other electronic signature related data from the M server and returns it in a HashMap. The key values in the HashMap are:

- initial
- signature block printed name
- signature block title
- office phone
- voice pager
- digital pager

```
public static final java.util.HashMap getESigData
    (gov.va.med.vistalink.adapter.cci.VistaLinkConnection connection)

    throws ESigConnectionException,
        ESigNotAValidUserException
```

Parameters:
   `connection` - The `VistaLinkConnection` handle.

Returns:
   A HashMap that contains the user's data.

Throws:
   `ESigConnectionException` - if the RPC request fails.
   `ESigNotAValidUserException` - if the DUZ of the user does not correspond to a valid entry in the New Person file.

Example:

```
Map oldValues = null;
try {
    oldValues = ESigDataAccess.getESigData(myConnection);
    System.out.println("Values of Map returned:");
    System.out.println("                      INITIAL: " +
        oldValues.get("initial"));
    System.out.println("SIGNATURE BLOCK PRINTED NAME: " +
        oldValues.get("signature block printed name"));
```

```
        System.out.println("        SIGNATURE BLOCK TITLE: " +
           oldValues.get("signature block title"));
        System.out.println("                  OFFICE PHONE: " +
           oldValues.get("office phone"));
        System.out.println("                   VOICE PAGER: " +
           oldValues.get("voice pager"));
        System.out.println("                 DIGITAL PAGER: " +
           oldValues.get("digital pager"));
    } catch (Exception e) {
      System.out.println(e.getMessage());
    }
```

saveESigData

The `saveESigData` method accepts the following values in a Map, and saves the values in the New Person file on the M server:

- initial
- signature block printed name
- signature block title
- office phone
- voice pager
- digital pager

```
public static final void saveESigData(java.util.Map values,
   gov.va.med.vistalink.adapter.cci.VistaLinkConnection connection)
      throws ESigConnectionException, ESigNotAValidUserException,
      ESigInvalidFormatException
```

Parameters:
    `values` - The values passed in a HashMap
    `connection` - The VistaLinkConnection handle.

Throws:
    `ESigConnectionException` - if the RPC request fails.
    `ESigNotAValidUserException` - if the DUZ of the user does not correspond to a valid entry in the New Person file.
    `ESigInvalidFormatException` - if the format of any of the data passed in is not valid. In this case, none of the data is filed.

Example:

```
HashMap newValues = new HashMap();
newValues.put("initial", "TAS");
newValues.put("signature block printed name", "Test A. Smith");
newValues.put("signature block title", "Dietician");
newValues.put("office phone", "(123) 123-4567");
newValues.put("voice pager", "(234) 234-5678");
```

```
newValues.put("digital pager", "(345) 345-6789");
try {
   ESigDataAccess.saveESigData(newValues, myConnection);
   LOGGER.info("New values saved successfully.");
} catch (Exception e) {
   System.out.println(e.getMessage());
}
```

2.2.2 Class ESigEncryption

The `ESigEncryption` class contains static methods to provide the checksum calculation. This class also implements encoding and decoding APIs comparable to those provided in VA Kernel v.8.0.

checksum

The `checksum` method calculates a checksum number for a String using the same algorithm as the Kernel `$$CHKSUM^XUSESIG1` function.

```
public static final java.lang.String checksum
   (java.lang.String document)
```

Parameters:
> `document` - A String containing the document for which to calculate a checksum value.

Returns:
> The checksum value.

Example:

```
String aDocument = "This is a sample document.\nA second line.\n";
String checksum = ESigEncryption.checksum(aDocument);
System.out.println("   Java checksum: " + checksum);
System.out.println("       aDocument:\n" + aDocument);
```

encrypt

This method encrypts a String using the same algorithm as the Kernel `EN^XUSHSHP` entry point.

```
public static final java.lang.String encrypt(java.lang.String text,
   double idNumber, double docNumber)
```

Parameters:
> `text` - The String to be encrypted.
> `idNumber` - An identification number, such as DUZ.
> `docNumber` - A document number (or the number one).

Returns:
    The encrypted version of the input String.

Example:

```
String aStringToEncrypt = "John A. Smith, MD";
double id = 101.0;
double doc = 53684791;
String encryptedText = ESigEncryption.encrypt(aStringToEncrypt, id,
   doc);
System.out.println("                   aString: " + aStringToEncrypt);
System.out.println("  Java encrypted value: " + encryptedText);
```

decrypt

This method decrypts a String using the same algorithm as the Kernel DE^XUSHSHP entry point.

```
public static final java.lang.String decrypt(java.lang.String text,
   double idNumber, double docNumber)
```

Parameters:
    text - The String to be decrypted.
    idNumber - The identification number used as the idNumber input parameter to the encrypt call.
    docNumber - The document numbers used as the docNumber input parameter to the encrypt call.

Returns:
    The decrypted version of the input String.

Example:

```
String decryptedText = ESigEncryption.decrypt(encryptedText, id,
doc);
System.out.println("                   aString: " + encryptedText);
System.out.println("  Java decrypted value: " + decryptedText);
```

hash

This overloaded method hashes a String or characters in a character array using the same algorithm as the Kernel HASH^XUSHSHP entry point. This method is used to hash an electronic signature code entered by the user.

```
public static final java.lang.String hash(java.lang.String text)
```

```
public static final java.lang.String hash(char[] text)
```

Parameters:
> `text` - The text to hash, contained in a String or character array.

Returns:
> A String that is the hashed form of the text in the input array.

Example:

```
String aString = "AnESigForTesting";
String hashedText = ESigEncryption.hash(aString);
System.out.println("              aString: " + aString);
System.out.println("  Java hashed string: " + hashedText);
```

2.2.3 Class ESigValidation

The `ESigValidation` class contains static methods that validate a user-supplied electronic signature code.

isValid

The overloaded `isValid` method validates a user-supplied electronic signature code against the electronic signature code stored in the New Person file (#200) on the M server. It returns `true` if the electronic signature code passed matches the code stored on the M server.

A VistALink connection is assumed, and the `VistaLinkConnection` object must be passed to the method along with the electronic signature code being validated.

```
public static final boolean isValid(char[] code,
    gov.va.med.vistalink.adapter.cci.VistaLinkConnection connection)
        throws ESigConnectionException,
          ESigNotAValidUserException,
          ESigNoElectronicSignatureDefinedException

public static final boolean isValid(java.lang.String code,
    gov.va.med.vistalink.adapter.cci.VistaLinkConnection connection)
        throws ESigConnectionException,
          ESigNotAValidUserException,
          ESigNoElectronicSignatureDefinedException
```

Parameters:
> `code` - A character array or String containing the unencrypted user-supplied
> electronic signature code.
> `connection` - The `VistaLinkConnection` handle.

Returns:
> `true` if the electronic signature code matches the code store on the M server.

Throws:
> `ESigConnectionException` - if an error occurred while attempting to make an RPC call on the M server.
> `ESigNotAValidUserException` - if the user identified by DUZ on the M server does not correspond to an entry in the New Person file.
> `ESigNoElectronicSignatureDefinedException` - if the user has no electronic signature defined on the M server.

Example:

```
try {
    boolean valid = ESigValidation.isValid(userESig.toCharArray(),
        myConnection);
    if (valid) {
        System.out.println("Electronic signature code is valid.");
    } else {
        System.out.pritnln("Electronic signature is not valid.");
    }
} catch (ESigConnectionException e) {
    System.out.println(e.getMessage());
} catch (ESigNotAValidUserException e) {
    System.out.println(e.getMessage());
} catch (ESigNoElectronicSignatureDefinedException e) {
    System.out.println(e.getMessage());
}
```

isValidFormat

The overloaded `isValidFormat` method checks whether the format of the user-supplied electronic signature code is valid. Electronic signatures codes must be between 6 and 20 characters in length, and cannot contain control characters; that is, they must contain only the printable characters in the 7-bit ASCII character set, decimal ASCII values 32 through 126.

```
public static final boolean isValidFormat(char[] code)

public static final boolean isValidFormat(java.lang.String code)
```

Parameters:
> `code` - A character array containing the unencrypted user-supplied electronic signature code.

Returns:
> `true` if the format of the electronic signature code is valid.

Example:

```
String[] validESigCodes =
    { "6CHARS",
    "LENGTH 20 CHARACTERS",
    "`~!@#$%^&*()-_=+",
    "[]\\{}|;:'\",./<>?",
    "VALID_INCL.PUNC" };
String[] invalidESigCodes =
    { "SHORT", "", "THIS ELECTRONIC SIGNATURE IS TOO LONG", "Invalid
    mixed case", };

System.out.println(" Valid e-sig codes:");
for (int i = 0; i < validESigCodes.length; i++) {
    System.out.println("   " + validESigCodes[i]);
    System.out.println("      --> " +
        (ESigValidation.isValidFormat(validESigCodes[i]) ? "valid" :
        "invalid"));
}

System.out.println("");
System.out.println(" Invalid e-sig codes:");
for (int i = 0; i < invalidESigCodes.length; i++) {
    if (invalidESigCodes[i].equals("")) {
        System.out.println("   <null> string");
    } else {
        System.out.println("   " + invalidESigCodes[i]);
    }
    System.out.println("      --> " +
        (ESigValidation.isValidFormat(invalidESigCodes[i]) ? "valid" :
        "invalid"));
}
```

3 ESig Exception Hierarchy

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

ESigException is the super class of the more specific exceptions thrown by the electronic signature code utilities. When using the utilities, you can catch the more general ESigException, in addition to the specific exceptions that may be thrown.

The following is the inheritance hierarchy of the Exceptions used by ESig:

```
java.lang.Object
    java.lang.Throwable
        java.lang.Exception
            gov.va.med.exception.FoundationsException
                gov.va.med.esig.utilities.ESigException

                    ESigConnectionException
                    ESigInvalidFormatException
                    ESigNoElectronicSignatureDefinedException
                    ESigNotAValidUserException
                    ESigUnchangedElectronicSignatureException
```

ESig Exception Hierarchy

4 Using the ESig Sample Applications

Once ESig has been successfully installed on the VistA/M server, you can use the Electronic Signature sample applications to verify that everything is installed correctly on the VistA/M server. The sample applications also exercise the ESig APIs and demonstrate how they can be used.

Three sample applications are included in the distribution:

- A command-line J2SE sample application.
- A Swing J2SE sample application.
- A J2EE sample application, which is deployed to the J2EE application server.

4.1   Prerequisites

The sample applications are run from the application server (J2EE) and/or the client workstation (J2SE) to exercise the ESig APIs.

4.1.1 User Requirements

Developers and testers of applications using ESig may need to set up M and Java environments to test ESig functionality. Some familiarity with the following areas is required to do this:

- VistA computing environment
- Installing software and managing a VistA/M system
- Setting up a VistALink 1.5 listener and confirming that the service is enabled
- Kernel Installation and Distribution System (KIDS)
- VA FileMan data structures and terminology
- Microsoft Windows
- Red Hat Linux
- M programming language
- Deploying software and managing a J2EE application server
- Configuring and managing a server cluster.

4.1.2 Client Workstation Requirements

Applications that use Electronic Signature on a client workstation require the hardware and software tools listed below.

4.1.2.1 <u>Hardware</u>

The client workstation is the only hardware requirement.

4.1.2.2 <u>Operating System</u>

Microsoft Windows 2000/XP

4.1.2.3   Network Communications Software

Electronic Signature requires networked client workstations running Microsoft's native TCP/IP stack.

4.1.3   J2EE System Requirements

4.1.3.1   Application Server

BEA WebLogic Server 8.1, service pack 4 or greater. (The ESig J2EE sample application has been tested only on WebLogic at this time.)

4.1.3.2   Operating System

Windows or Linux as the platform operating system for WebLogic Server

Electronic Signature has not been tested and is not supported on BEA WebLogic Server 9.x

4.2   J2SE Sample Applications

Two J2SE (client/server) applications are included in the **samples\J2SE** directory of the Electronic Signature distribution:

- **runEsigSample.bat** - A sample command-line application that runs in a DOS window.
- **runEsigApiSwingTester.bat** - A sample Swing application.

Both sample applications make calls to all the ESig APIs. Some of these calls use VistALink to connect to the VistA/M server and invoke RPCs installed via the KIDS build.

Before running the sample applications, you will need to deploy them on the client workstation and reconfigure three of the ESig distribution files. This section presents instructions for deploying and running the sample applications and reconfiguring these files.

4.2.1   Required Java Software

For the ESig J2SE sample application to work properly, you must have the proper environment and supporting libraries already installed on the client workstation, as follows:

- **J2SE Java Runtime Environment (JRE) 1.4.2**

  The complete Java 2 Standard Edition (J2SE) environment, version 1.4.2 or higher, must be installed on the client workstation. You can obtain the complete J2SE environment from http://java.sun.com/.

- **Required Supporting Libraries**

  The libraries listed in the table below are required by VistALink 1.5, and therefore must also be on the classpath of applications using ESig.

Table 4-1. Required Electronic Signature Supporting Libraries

Library	Minimum Version	JAR File Name and Description	Obtain From
J2EE core library	1.3.1	**j2ee.jar** Part of the Java 2 Enterprise Edition (J2EE) Standard Development Kit (SDK)	http://java.sun.com
Jaxen	1.0FCS	**jaxen-full.jar** Java XPath engine	http://sourceforge.net/projects/jaxen/
SAXPath	1.0FCS	**saxpath.jar** Simple API for XPath	Included in Jaxen distribution
JAXP-Compatible XML Parser	JAXP 1.1	(various) Any XML parser that implements the JAXP interface. For example: • Xerces • Crimson • Oracle XDK	**Xerces**: Included in Jaxen (xerces.jar) **Crimson**: http://xml.apache.org/crimson/ **Oracle XDK**: http://technet.oracle.com/tech/xml/
JAXP-Compatible XSLT Processor	JAXP 1.1	(various) Any XSLT processor that implements the JAXP interface. For example: • Xalan-Java • Saxon	**Xalan-Java**: included in j2ee.jar (v. 1.3.1), or http://xml.apache.org/xalan-j/ **Saxon**: http://saxon.sourceforge.net/
Log4J	1.2.8	**log4j-x.x.x.jar** Java Logging Utility.	http://jakarta.apache.org/log4j/docs/
VistALink	1.5	vljConnector-1.5.x.nnn.jar vljFoundationsLib-1.5.x.nnn.jar vljSecurity-1.5.x.nnn.jar	Office of Information ANONYMOUS directories (XOB* namespace).

4.2.2 Deploying the Sample Apps

4.2.2.1 <u>Unzip the Distribution File</u>

Unzip the Electronic Signature distribution file (XOBE_1.0.zip) to a directory of your choice on the client workstation (e.g., **c:\esig-1.0.0.xxx**). The directory in which you extract the zip file is referred to below as <DISTRIBUTION_HOME>.

4.2.2.2 <u>Confirm Distribution Files</u>

You need the distribution files listed in the table below to run the sample Electronic Signature J2SE applications on the client workstation:

Table 4-2. ESig Installation Distribution Files for Client Workstation

File Name	Location	Description
esig-1.0.0.xxx.jar	<DISTRIBUTION_HOME>\jars <DISTRIBUTION_HOME>\samples\J2SE	Contains the Electronic Signature library classes.
esigSamples-1.0.0.xxx.jar	<DISTRIBUTION_HOME>\jars <DISTRIBUTION_HOME>\samples\J2SE	Contains the classes for the J2SE sample application.
jaas.config	<DISTRIBUTION_HOME>\samples\J2SE	A JAAS configuration file used by the sample console and Swing applications to connect to the M server. You must modify this file to reflect your environment.
runESigApiSwingTester.bat	<DISTRIBUTION_HOME>\samples\J2SE	Runs the sample Swing application. Uses the settings defined in setEsigEnvironment.bat and jaas.config.
runESigSample.bat	<DISTRIBUTION_HOME>\samples\J2SE	Runs the sample console application. You must modify this file to specify the server alias, an access and verify code pair, and a division IEN for a valid user.
setESigEnvironment.bat	<DISTRIBUTION_HOME>\samples\J2SE	Sets the JAVA_HOME and CLASSPATH environment variables to reflect the name and locations of the java.exe executable and the JAR files used by the sample console and Swing applications. You must modify this file reflect your environment.

File Name	Location	Description
jaxen_core.jar jaxen_dom.jar log4j-1.2.8.jar saxpath.jar	<DISTRIBUTION_HOME>\samples \J2SE	JARs required by Electronic Signature and/or VistALink 1.5.

4.2.2.3   Create Samples Directory

Create a directory to hold the sample ESIG application files (e.g., C:\Program Files\esig-1.0\samples). This directory will be referred to "<ESIG_SAMPLE_APP>."

4.2.2.4   Copy J2SE Samples Folder

Copy the contents of the **<DISTRIBUTION_HOME>\samples\J2SE** folder to the **<ESIG_SAMPLE_APP>** directory.

4.2.2.5   Copy Required JARs

ESig requires specific supporting libraries on the client workstation.

1. You need either **weblogic.jar** or **j2ee.jar**. Do one of the following:

    -   Download and install the 1.3.x J2EE SDK (http://java.sun.com/j2ee/sdk_1.3/), to obtain the **j2ee.jar** file. (The SDK can then be un-installed.)

    -   If you have access to an installed WebLogic server, you can use the **weblogic.jar** file from the WebLogic server installation directory's **lib** subdirectory.

2. Copy **j2ee.jar** or **weblogic.jar** to the <ESIG_SAMPLE_APP> directory.

3. Copy the VistALink 1.5 JAR files:

    vljConnector-1.5.x.nnn.jar
    vljFoundationsLib-1.5.x.nnn.jar
    vljSecurity-1.5.x.nnn.jar

    to the **<ESIG_SAMPLE_APP>** directory.

4.2.3   Configuring ESig Files

Before you can run the sample ESig J2SE applications, the following three files must be modified to reflect your environment:

    setESigEnviornment.bat
    jaas.config
    runESigSample.bat

4.2.3.1 Modify the setESigEnvironment.bat File

The **setESigEnvironment.bat** file sets the CLASSPATH and the JAVA_HOME environment variables. This batch file is called by the two sample application batch files. To configure the CLASSPATH and JAVA_HOME variables for both applications, you need to modify only this one file.

1. Locate the **setESigEnvironment.bat** file in the **<ESIG_SAMPLE_APP>** folder (e.g., c:\Program Files\esig-1.0\samples).

2. Use a text editor such as Notepad or WordPad to set the JAVA_HOME and CLASSPATH environment variables to reflect the location of the Java installation on the workstation and the names and locations of the JAR files used by Electronic Signature.

3. Save the file when you are finished.

Example:

```
REM -- Set the JAVA_HOME and the CLASSPATH environment variables.
REM
REM -- You will need to modify the following lines to match the names and
REM -- locations of the jars and other files on your system.
REM
REM -- Set the directory location containing java.exe executable.
REM -- (Don't include the \bin subdirectory.)
REM -- set JAVA_HOME=c:\program files\java\j2re1.4.2
set JAVA_HOME=c:/j2sdk1.4.2_12
REM
REM -- Set CLASSPATH for J2EE (j2ee.jar or weblogic.jar)
REM set CLASSPATH=./weblogic.jar
set CLASSPATH=./j2ee.jar
REM
REM -- classpath for XML libraries
set CLASSPATH=%CLASSPATH%;./jaxen-core.jar
set CLASSPATH=%CLASSPATH%;./jaxen-dom.jar
set CLASSPATH=%CLASSPATH%;./saxpath.jar
REM
REM -- classpath for log4j
set CLASSPATH=%CLASSPATH%;./log4j-1.2.8.jar
REM
REM -- class path for VistALink
REM --   Replace the following with the correct names of the VistALink 1.5 jars
REM --   you are using.
set CLASSPATH=%CLASSPATH%;./vljConnector-1.5.X.XXX.jar
set CLASSPATH=%CLASSPATH%;./vljFoundationsLib-1.5.X.XXX.jar
set CLASSPATH=%CLASSPATH%;./vljSecurity-1.5.X.XXX.jar
REM
REM -- classpath for Electronic Signature
set CLASSPATH=%CLASSPATH%;./esig-1.0.0.024.jar
REM
REM -- classpath for ESigSample app
REM --   (Replace 1.0.0.024.jar with the correct version #)
REM --   (assumes the samples jar is in the current directory)
set CLASSPATH=%CLASSPATH%;./esigSamples-1.0.0.024.jar
```

4.2.3.2 Modify the jaas.config File

1. Locate the **jaas.config** file in the <ESIG_SAMPLE_APP> folder (e.g., c:\Program Files\esig-1.0\samples).

2. Use a text editor to modify the `ServerAddressKey` and `ServerPortKey` fields to reflect the settings for connecting to your M system. The settings specify the IP address and port on which the VistALink listener is running.

   - ServerAddressKey = Host IP Address or DNS name
   - ServerPortKey = VistALink Listener Port Number

3. The **jaas.config** file contains two sample configuration entries: `DemoServer` and `LocalServer`. The **runESigSample.bat** file is hard-coded to load a configuration named "LocalServer" from the **jaas.config** file. You can do one of the following:

   - Modify the `LocalServer` configuration with the settings needed for your M system.

   - If you use a different configuration and configuration name, modify **runEsigSample.bat** to use your configuration name. (The configuration name is specified via the **-s** parameter at the end of the command line that launches the application.)

   For more information on defining login configurations in the **jaas.config** file, see Appendix A "Installing and Running J2SE Sample Apps" of the *VistALink 1.5 Installation Guide.*

Example:

```
DemoServer {
      gov.va.med.vistalink.security.VistaLoginModule requisite
            gov.va.med.vistalink.security.ServerAddressKey="localhost"
            gov.va.med.vistalink.security.ServerPortKey="8001";
};
LocalServer {
      gov.va.med.vistalink.security.VistaLoginModule requisite
            gov.va.med.vistalink.security.ServerAddressKey="127.0.0.1"
            gov.va.med.vistalink.security.ServerPortKey="8001";
};
```

Either the DNS name or an IP address may be used as the `ServerAddressKey` value.

Check with your VistA system manager for the VistALink port assignment on your system. While port 8000 is suggested for Production and 8001 for Test environments, any available port number may be assigned to the VistALink Listener(s).

4.2.3.3 <u>Modify the runESigSample.bat File</u>

1. Locate the **runESigSample.bat** file in the **<ESIG_SAMPLE_APP>** folder (e.g., c:\Program Files\esig-1.0\samples).

2. Use a text editor to modify the **-s**, **-a**, **-v**, and **-d** program arguments to reflect an appropriate server alias (configuration name from **jaas.config**), an access and verify code pair, and division IEN for a valid user.

3. Save the file when you are finished.

Example:

```
REM -- Runs the ESigSample application, in esigSamples-1.0.0.xxx.jar.
REM
REM -- Depends on variables CLASSPATH and JAVA_HOME, both set in
REM -- setESigEnvironment.bat.
REM
REM -- You will need to adjust the locations of the various jars and
other files
REM -- in setESigEnvironment.bat to match the locations of these files
on your
REM -- system.
REM
call setESigEnvironment.bat
REM
REM -- Run the sample application. (Assumes the jaas.config file is in
the
REM -- current directory.)
REM -- You will need to provide values for the following flags:
REM --    -s server alias to use from the JAAS config file
REM --    -a access code
REM --    -v verify code
REM --    -d division ien (if more than one division can be selected)
REM -- For example:
REM --    java -Djava.security.auth.login.config=./jaas.config
VistaLinkRpcConsole
REM --    -s MyServer -a ac!@#$12 -v vc123!@# -d 999
"%JAVA_HOME%\bin\java" -Djava.security.auth.login.config="./jaas.config"
-cp "%CLASSPATH%" gov.va.med.esig.samples.ESigSample -s LocalServer -a
joe.123 -v ebony.432 -d 999
pause
```

4.2.4 Accessing ESig Remote Procedures

The Kernel "B"-type option, **Context for Electronic Signature Users [XOBE ESIG USER]**, was created as part of the M-side KIDS install. To use the ESig APIs and to run the sample ESig applications, you will need to grant yourself access to [XOBE ESIG USER] on the VistA/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 remote procedures, see the *RPC Broker Systems Manual* at http://www.va.gov/vdl/.

4.2.5 Running the Sample Applications

4.2.5.1 <u>Check that the VistALink Service is Enabled</u>

In a production scenario, VistALink is configured as a TCP/IP service in VMS. Here is an example of the service (VLINK) in its enabled state:

```
01$TCPIP
TCPIP> SHO SERVICE VLINK /FULL


Service: VLINK                         State:  Enabled
Port:       8000    Protocol:  TCP              Address:  0.0.0.0
Inactivity:     1    User_name: XMINET        Process:  VLINK
Limit:        50    Active:     0               Peak:     2
```

4.2.5.2 <u>Run the Electronic Signature Sample Console Application:</u>

1. Locate the **runESigSample.bat** in the **<ESIG_SAMPLE_APP>** folder (e.g., c:\Program Files\esig-1.0\samples).

2. Double-click the file to launch the sample application and to test your installation of Electronic Signature.

Here is a portion of the output from executing the runESigSample.bat file:

```
Starting sample application...

This application defaults to using the following connection values:

    Server appname: LocalServer
    Access code: access.1234
    Verify code: verify.1234

 Override defaults with the following optional flags:

    -s server alias to use from the JAAS config file
    -a access code
    -v verify code
    -d division ien

e.g.: java -Djava.security.auth.login.config=./jaas.config
VistaLinkRpcConsole -
s MyServer -a ac!@#$12 -v vc123!@#

Also, for Log4J initialization, the Log4J config file
'log4JConfig.xml' is expec
ted to be in the classpath location
```

```
props/log4jConfig.xml

Logging in...
JAAS configuration name: LocalServer
93   [main] INFO   Creating managed connection factory, VistALink
adapter versio
n 1.5.0.026.
93   [main] WARN   gov.va.med.environment.servertype is not defined
for this JVM
. For J2EE systems only -- check this -D JVM arg; MBeans required for
the VistaL
ink console will not be loaded.
281  [main] INFO   Socket xfer (milli-secs): 16
281  [main] INFO
gov.va.med.vistalink.adapter.spi.VistaLinkManagedConnection[]
127.0.0.1[]8001[]1[]J2SE[fdi]1[mdi]1 M $JOB=3292
          getConnection(...) processing time = 0
3421 [main] INFO   Socket xfer (milli-secs): 3125
6593 [main] INFO   Socket xfer (milli-secs): 3156
9968 [main] INFO   Socket xfer (milli-secs): 15


=============================================================
Calling ESigEncryption.hash(aString)
             aString: AnESigForTesting
   Java hashed string: {]nYPg_u;G)p<rcN]/WC
=============================================================
9999 [main] INFO   Socket xfer (milli-secs): 0


=============================================================
Calling ESigEncryption.checksum(aDocument)
   Java checksum: 53684791A
       aDocument:
==================
Gettysburg Address
==================
Four score and seven years ago our fathers brought forth,
upon this continent, a new nation, conceived in liberty,
and dedicated to the proposition that 'all men are
created equal'.

Now we are engaged in a great civil war, testing whether that
nation, or any nation so conceived, and so dedicated, can long
endure. We are met on a great battle field of that war. We come
to dedicate a portion of it, as a final resting place for those
who died here, that the nation might live. This we may, in all
propriety do. But, in a larger sense, we can not dedicate -- we
can not consecrate -- we can not hallow, this ground -- The brave
men, living and dead, who struggled here, have hallowed it, far
above our poor power to add or detract. The world will little
note,nor long remember what we say here; while it can never
forget what they did here."

It is rather for us, the living, we here be dedicated to the great
task remaining before us -- that, from these honored dead we take
increased devotion to that cause for which they here, gave the
last full measure of devotion -- that we here highly resolve these
```

```
dead shall not have died in vain; that the nation, shall have a
new birth of freedom, and that government of the people by the
people for the people, shall not perish from the earth.

============================================================


============================================================
Calling ESigEncryption.encrypt(aString, 101.0, 5.3684791E7)
              aString: John A. Smith, MD
   Java encrypted value: YRV24~drC#V6xN"m$
============================================================


============================================================
Calling ESigEncryption.decrypt(aString, 101.0, 5.3684791E7)
              aString: YRV24~drC#V6xN"m$
   Java decrypted value: John A. Smith, MD
============================================================


============================================================
Calling ESigValidation.isValid(aString, VistaLinkConnection)
  E-sig code being validated: MY ESIG CODE
10031 [main] INFO   Socket xfer (milli-secs): 16
Electronic signature code is valid.
============================================================


============================================================
Calling ESigValidation.isValidFormat(String)
 Valid e-sig codes:
   6CHARS
     --> valid
   LENGTH 20 CHARACTERS
     --> valid
   `~!@#$%^&*()-_=+
     --> valid
   []\{}|;:'",./<>?
     --> valid
   VALID_INCL.PUNC
     --> valid

 Invalid e-sig codes:
   SHORT
     --> invalid
   <null> string
     --> invalid
   THIS ELECTRONIC SIGNATURE IS TOO LONG
     --> invalid
   Invalid mixed case
     --> valid
============================================================


============================================================
Calling ESigDataAccess.getESigCode(VistaLinkConnection)
10140 [main] INFO   Socket xfer (milli-secs): 16
  ESig obtained from VistA: !JqN[Ww:HN&J,(MT/qeJ
============================================================
```

```
============================================================
Calling ESigDataAccess.saveESigCode(String, VistaLinkConnection)
Value attempting to save: NEW ESIG VALUE
10171 [main] INFO    Socket xfer (milli-secs): 15
Value NEW ESIG VALUE saved successfully.

Value attempting to save: MY ESIG CODE
10187 [main] INFO    Socket xfer (milli-secs): 16
Value MY ESIG CODE saved successfully.
============================================================


============================================================
Calling ESigDataAccess.getESigData(VistaLinkConnection)
10203 [main] INFO    Socket xfer (milli-secs): 16
Values of Map returned:
                     INITIAL: tat
SIGNATURE BLOCK PRINTED NAME: Mr. Test A. Testing
       SIGNATURE BLOCK TITLE:
                OFFICE PHONE: (415) 111-2222
                 VOICE PAGER:
               DIGITAL PAGER:
============================================================


============================================================
Calling ESigDataAccess.saveESigData(Map, VistaLinkConnection)
Attempting to save new values:
                     INITIAL: TAz
SIGNATURE BLOCK PRINTED NAME: Test A. TEST
       SIGNATURE BLOCK TITLE: Dietician
                OFFICE PHONE: (123) 123-4567
                 VOICE PAGER: (234) 234-5678
               DIGITAL PAGER: (345) 345-6789
10218 [main] INFO    Socket xfer (milli-secs): 15
New values saved successfully.

Attempting to save original values:
                     INITIAL: tat
SIGNATURE BLOCK PRINTED NAME: Mr. Test A. Testing
       SIGNATURE BLOCK TITLE:
                OFFICE PHONE: (415) 111-2222
                 VOICE PAGER:
               DIGITAL PAGER:
Original values saved successfully.
10249 [main] INFO    Socket xfer (milli-secs): 15
============================================================
Logging out...
10249 [main] INFO    Socket xfer (milli-secs): 0
10265 [main] INFO    Socket xfer (milli-secs): 0
```

4.2.5.3 Run the Electronic Signature Sample Swing Application:

1. Locate **the runESigApiSwingTester.bat** in the <ESIG_SAMPLE_APP> folder (e.g., c:\Program Files\esig-1.0\samples).

2. Double-click the file to launch the sample J2SE application and test your installation of Electronic Signature.

3. Select the name of the configuration (obtained from **jaas.config**) of the VistA/M server you wish to connect to and enter your VistA Access and Verify codes. A successful login will render a screen capture similar to the one shown below:
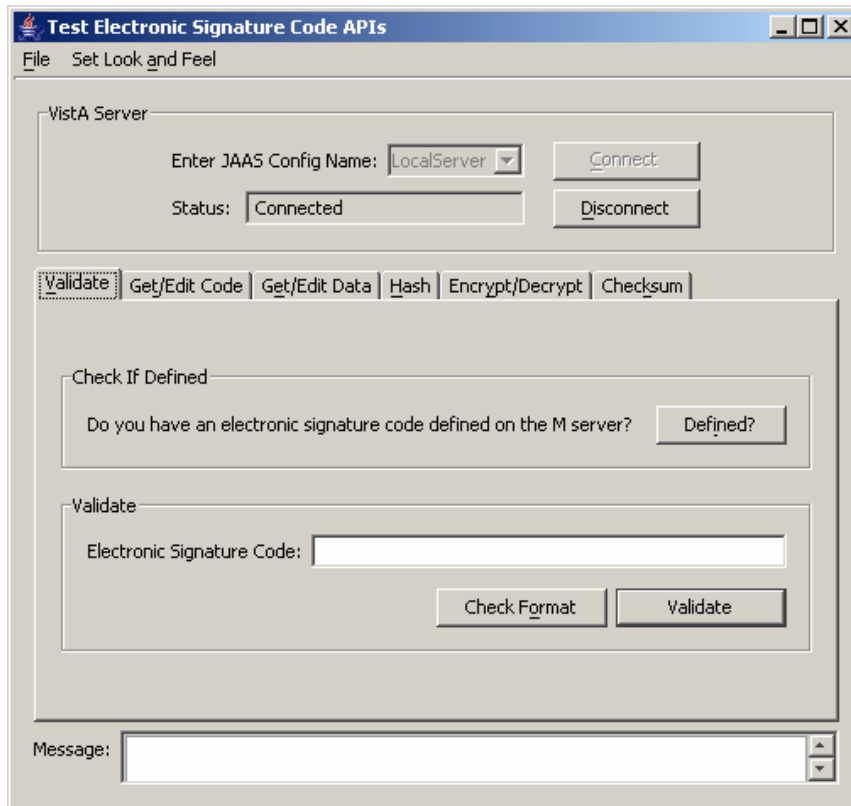

Figure 4-1. Successful Sample J2SE Swing Application Login

4.3   Sample J2EE Application

The **samples\J2EE** directory in the ESig 1.0 distribution contains a sample JSP application that does the following:

- Uses VistALink 1.5 (and the DUZ re-authentication mechanism) to connect to your VistA/M server
- Makes Electronic Signature API calls.

The enterprise application archive file, **EsigDuzSample-1.0.0.nnn.ear** (or the exploded EAR) can be deployed to the J2EE application server. The default Web binding (URL) is **/EsigDuzSample**, and is provided in the EAR file. The same application is packaged in a Web application archive file, **EsigDuzSample-1.0.0.nnn.war,** which can be used if your application server supports web application deployments.

The sample application requires a VistALink adapter with the JNDI lookup name of "vlj/testconnector." The adapter is deployed on the application server where the ESig sample application is installed.

ℹ️ For information on configuring **vlj/testconnector**, see "Testing the Sample Application with Your Own VistA/M Server" in the *VistALink 1.5 Installation Guide.*

4.3.1 Deploying the J2EE Sample App

The sample application is included in the Electronic Signature distribution zip file, in the **<DIST FOLDER>/samples/J2EE** folder. Both packaged and exploded EAR formats are provided. Follow the steps below to deploy the sample application on the application server:

1. Copy either the packaged EAR or the contents of the **<DIST FOLDER>/samples/J2EE/exploded folder** to the <STAGING FOLDER>.

    ℹ️ <STAGING FOLDER> designates a folder somewhere on the file system of each WebLogic server from which you are deploying the sample application (e.g., /bea-stage).

2. Use the WLS console to deploy either the packaged or exploded EAR (via the **<domain name> | Deployments | Applications** node):

    a. Select **Deploy a new Application**

    b. Navigate to the location where you copied the packaged or exploded EAR file (e.g., <STAGING FOLDER>)

    c. Select the radio button for the packaged or exploded EAR, and deploy the application to the target server.

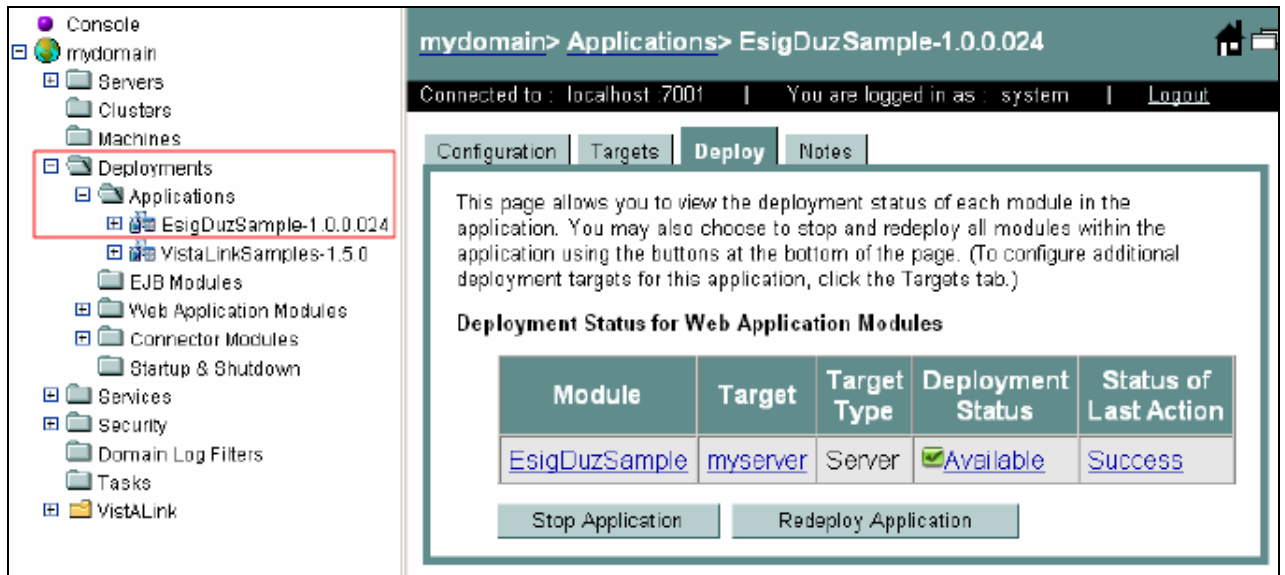3. **EsigDuzSample** will be listed under the applications node in the WLS console (see the figure below).

Figure 4-2. ESIG Exploded EAR deployed to WLS

4.3.2 Running the J2EE Sample App

The sample J2EE application can be run on either Linux or Windows, and on either administration or managed servers. Follow these steps to run the sample J2EE sample application:

1. Point your browser to http://<yourserver:yourport>/EsigDuzSample.

    If the deployment was successful, a new page titled "Electronic Signature Test – DUZ Logon" will load in your browser.

2. Enter a DUZ and a division number (e.g., station number) for a user on the VistA/M system to which the **vlj/testconnector** resource adapter is pointed.

    Note: Since the Electronic Signature code of this user will be changed by the sample application, the user selected should be a test user.

3. Click on the **Submit** button.

    The figure below shows an example of an end user's DUZ and division entry.

Figure 4-3. Electronic Signature Login Page

The sample application will attempt to make Electronic Signature API calls. A successful installation and deployment, with a valid DUZ and division entry for the logged in user, will result in a display similar to the two figures below.

Electronic Signature Sample - Results

Report Generated on Tue Nov 22 14:32:14 PST 2005.

Credentials used:

Duz : 11668

Division : 662

Testing ESig APIs that do not connect to the VistA/M Server

Calling `ESigEncryption.hash(esigString)`
esigString: **A String for hashing**
Java hashed string: (+2l2W|4RFd9thlzWnG$

Calling `ESigEncryption.encrypt(aString, 101.0, 5.3684791E7)`
aString: **John A. Smith, MD**
Java encrypted value: YRV24~drC#V6xN"m$

Calling `ESigEncryption.decrypt(aString, 101.0, 5.3684791E7)`
aString: YRV24~drC#V6xN"m$
Java decrypted value: **John A. Smith, MD**

Calling `ESigEncryption.checksum(aString)`
aString: **ADocumentForTesting**
Java checksum: **14008A**

Calling `ESigValidation.isValidFormat(String)`
Valid e-sig codes:
6CHARS --> valid
LENGTH 20 CHARACTERS --> valid
` ~!@#$%^&*()-_=+ --> valid
[]\{}|;:'",./<>? --> valid
VALID_INCL.PUNC --> valid
Mixed Case --> valid

Invalid e-sig codes:
SHORT --> invalid
'null' string --> invalid
THIS ELECTRONIC SIGNATURE IS TOO LONG --> invalid

Figure 4-4. Electronic Signature Sample J2EE Application (top)

Testing ESig APIs that use VistALink to connect to the VistA/M Server

---

Calling `ESigDataAccess.isDefined(VistaLinkConnection)`
User has an e-sig defined: **true**

Calling `ESigValidation.isValid(aString, VistaLinkConnection)`
E-sig code being validated: **MY ESIG CODE**
Electronic signature code is valid.

Calling `ESigDataAccess.getESigCode(VistaLinkConnection)`
ESig obtained from VistA: **!JqN[Ww:HN&J,(MT/qeJ**

Calling `ESigDataAccess.saveESigCode(String, VistaLinkConnection)`
Value attempting to save: **MY NEW ESIG CODE**
Value **MY NEW ESIG CODE** saved successfully.
Note: The user's electronic signature code was changed to the above value.

Value attempting to save: **MY ESIG CODE**
Value **MY ESIG CODE** saved successfully.
Note: The user's electronic signature code was changed to the above value.

Calling `ESigDataAccess.getESigData(VistaLinkConnection)`
Values of Map returned:
INITIAL: **SU**
SIGNATURE BLOCK PRINTED NAME: **SAMPLE USERONE**
SIGNATURE BLOCK TITLE:
OFFICE PHONE:
VOICE PAGER:
DIGITAL PAGER:

Calling `ESigDataAccess.saveESigData(Map, VistaLinkConnection)`
Attempting to save new values:
INITIAL: **TAz**
SIGNATURE BLOCK PRINTED NAME: **Changed SAMPLE USERONE**
SIGNATURE BLOCK TITLE: **Dietician**
OFFICE PHONE: **(123) 123-4567**
VOICE PAGER: **(234) 234-5678**
DIGITAL PAGER: **(345) 345-6789**
New values saved successfully.

Attempting to save original values:
INITIAL: **SU**
SIGNATURE BLOCK PRINTED NAME: **SAMPLE USERONE**
SIGNATURE BLOCK TITLE:
OFFICE PHONE:
VOICE PAGER:
DIGITAL PAGER:
Original values saved successfully.

Figure 4-5. Electronic Signature Sample J2EE Application (bottom)

Glossary

Term	Definition
Access Code	A code, that along with the Verify code allows the Kernel to identify a user as authorized to gain access to a VistA system.
API	Application Programming Interface. The set of public classes a package uses. Intended to prevent duplication of utilities and limit the number of callable entry points.
ASCII	American Standard Code for Information Interchange
Authentication	Verification of a user's identity.
Authorization	Checking the permissions of a user to allow or disallow the performance of some function.
CCE	Computer Code Entry. A password/PIN technology for asserting electronic signature intent in a health-care environment. Computer Code Entry (CCE) is explicitly endorsed in existing medical records practice/regulation and is permitted by JCAHO IM7 standards.
Client	A single term used interchangeably to refer to a user, the workstation (e.g., PC), and the portion of the software that runs on the workstation.
Data Dictionary	The structure of a file, table, or group of related information as defined for and by VA FileMan
Database Integration Agreement (DBIA)	A formal, documented understanding between two or more application packages that describes how data is shared or information is exchanged. The Database Administrator (DBA) maintains these agreements. Documented agreements are available via the DBIA menu on FORUM
DBA	Data Base Administrator
Decrypt	To decipher, decode, or unlock encrypted or encoded messages/data to make them readable.
DUZ	DUZ represents the internal entry number (IEN) for a user's record in File #200, the New Person file and is designated as a system-wide variable in the VistA environment. DUZ is used as a re-authentication mechanism.
EAR	Enterprise ARchive file.
EJB	Enterprise Java Bean.
Electronic Signature	A secret, user-supplied PIN or code that is used to authorize business processes and is a legally binding equivalent of an individual's handwritten signature. For the VA Kernel 8.0 an electronic signature must be 6-20 characters in length and can contain letters, numbers, and punctuation.
Encrypt	To encode messages or data so that they are unreadable unless they are decoded.
ESig	Electronic Signature.
EVS	Enterprise VistA Support
FOIA	Freedom of Information Act
FTP	File Transfer Protocol
GUI	Graphical User Interface. The graphical elements on the screen through which the user interacts with the computer.
Hash	To encrypt data by substituting a shorter fixed-length value or key to represent the original. Hashing algorithms are one-way functions, so that it is not possible to decrypt the substitute values to generate the original data.
IP	Internet Protocol

Glossary

Term	Definition
IRM	Information Resources Management. A service at each VAMC responsible for computer management and system security.
ISO	Information Security Officer
J2EE	Java ^{TM} 2 Platform, Enterprise Edition
J2SE	Java 2 Standard Edition
JAAS	Java Authentication and Authorization Service
Javadoc	Javadoc is the tool from Sun Microsystems for generating API documentation in HTML format from doc comments in Java source code.
JNDI	Java Naming Directory Interface
JSP	Java Server Page
JVM	Java Virtual Machine
Kernel	VA Kernel 8.0 is a suite of VistA software that provides a standard and consistent user and programmer interface between application packages, the OS, and users.
M	MUMPS
Option	A selectable software function: a menu item.
PIN	Personal Identification Number
PKI	Public Key Infrastructure
RPC	Remote Procedure Call
SAC	Standards and Conventions
SACC	Standards and Conventions Committee
SDK	Java Software Development Kit. APIs and tools for developing applications.
Signature Block	Data associated with an electronic signature user, stored in the New Person file. Signature block data consists of the user's initials, printed name, title, office phone, voice pager, and digital pager.
TBD	To Be Determined
TCP/IP	Transmission Control Protocol / Internet Protocol
URL	Uniform Resource Locator
User	This term generally refers to VA employees and volunteers with active records established in File #200, the New Person file, who are authorized to access a VistA system.
VA	Veterans Affairs
VA ITSCAP	VA Information Technology Security Certification and Accreditation Program (VA Directive 6214)
VAMC	Department of Veterans Affairs Medical Center
VAX	VAX (Virtual Address eXtension) is an established line of mid-range server computers from the Digital Equipment Corporation (DEC).
VHA	Veterans Health Administration
VistA	Veterans Health Information Systems and Technology Architecture
VistA/M Server	The computer where the M data and the RPC Broker remote procedure calls (RPCs) reside.
VistALink	A standardized, portable, and secure mechanism for establishing synchronous connections between Java (J2SE and J2EE) and VistA/M servers.

Term	Definition
VMS	Virtual Machine System (operating system for VAX computers)
WAN	Wide Area Network
WAR	Web ARchive
WLS	WebLogic Server
XML	Extensible Markup Language