ORACLE

Oracle Forms Standalone Launcher (FSAL)

Running Applications with FSAL

February, 2023, Version 12.2.1.19 - Rev 0

Purpose statement

This document will outline the intended use for Oracle Forms Standalone Launcher (FSAL). Security information and suggestions provided in this document are for example purposes only. It is your responsibility to understand the concepts outlined and properly test and secure any implementation derived from this content. Misuse of the Oracle products or dependent technologies may result in the degradation of your system or application security. Be sure to refer to the included references to ensure you are properly using the product and any dependent technologies.

Table of contents

Purpose statement **2**

Introduction **4**

Understanding the Forms Standalone Launcher **5**

Requirements **5**

The Forms Standalone Launcher 5

Java 6

Java Runtime Environment - JRE 6

Java Development Kit - JDK 6

Server JRE 6

Running an Application with FSAL **7**

The Basics 7

File Caching 8

Running Audio Enabled Applications with Java 11+ 9

Using Java Script Integration with FSAL Applications 9

Running Through a Proxy Server 9

Launching FSAL with a Custom Protocol Handler 10

Security Tips **11**

Secure Socket Layer 11

Configuring SSL/TLS with FSAL 12

Using the FSAL Certificate Importer 12

Configuring the KeyStore Manually 13

Troubleshooting SSL/TLS 15

Signed Code 16

Forms, Java Plug-in, and Java Web Start 16

FSAL and Signed JARs 16

Single Sign-on 17

FSAL References **18**

FSAL Web Configuration Parameters 18

FSAL Command Line Arguments 19

Resources **21**

Introduction

As web browser software moves away from supporting plug-ins, technologies like Oracle Forms needed an alternative way in which to run the applications developed with them. Beginning in Forms version 12.2.1.0, one such alternative was introduced.

The Forms Standalone Launcher (FSAL) offers an alternative way for end-users to run Forms 12.2.1+ applications. FSAL provides a browser-less, client/server-like experience for users to access and interact with applications. As a result of not using a browser, FSAL does not rely on the Java Deployment technologies (e.g. Plug-in, Web Start, etc.) and therefore is not dependent on a browser. However, it does require the Java Runtime on the end-user machine.

This paper will offer FSAL usage information, as well as some tips on ensuring applications run as securely as possible.

This paper will assume the use of Oracle Forms 12.2.1.19.0. Some features described in this document may not be available in earlier releases of Forms 12c family; however many concepts will still apply to any 12.2.1 release. In the context of this paper, all Java references are assumed to be relative to Java version 8u341 or newer, unless noted otherwise.

Understanding the Forms Standalone Launcher

The Forms Standalone Launcher (FSAL) mimics several client/server concepts, however with some modern flare. In the early days of Oracle Forms client/server, end-user machines required the installation of the Oracle Forms runtime software, as well as the application(s) planned to be run on that machine. This was often a cumbersome process and consumed significant disk space. Having the software and application installed on the user machines was sometimes viewed as a security issue since it gave the user direct access to the runtime software, utilities included with it, and the compiled application(s) modules.

In the past, to make end-user machine administration easier, some organizations elected to not install the software locally and instead took advantage of remote installations and remote access across the network. Although the concept of remote access or remote desktop sharing worked in many cases, it too came with issues and was not (and still is not) consider a supported configuration by Oracle.

FSAL takes the best of both concepts; client/server and web. Using FSAL can provide the appearance and power of a natively installed application, very much like client/server Forms. The application runs in its own parent window, unlike when running a form in a browser. As a result, there is no risk of accidentally navigating away from the running form by pressing the Back or Forward button, or clicking a bookmark, etc. With FSAL, the application will be hosted on a centralized application server (e.g. WebLogic Server), just as in any web deployed Forms versions. This means the Forms application modules are securely stored on the remote middle tier server. Although the user will not have direct access to the Forms modules (e.g. FMX, MMX, PLX, etc.), which are on the remote server, they will be able to run these applications using a typical URL previously used when running in a browser.

The use of FSAL supports all the same functionality found when running a form in a browser, except event driven single sign-off. Single sign-on is supported beginning with 12.2.1.4.0. JavaScript integration support for applications launched using FSAL can be added/enabled with a provided add-on (WJSI) in 12.2.1.3.0+ and a third party library (Eclipse/Jetty). Beginning with 12.2.1.19.0, FSAL can be launched using custom protocol handlers from web pages.

Requirements

The Forms Standalone Launcher

As mentioned, the FSAL feature is only available in the Forms 12c family and newer. The configuration requires that a small Java JAR file (frmsal.jar) be transferred to and stored on the end-user machine. This file/utility can be transferred to the end-user machine using any desirable method (e.g. web download, email, ftp, custom installer, etc.). The utility can be stored anywhere on the user's machine as long the user has access to that directory and utility. Storing it in the user's home directory is recommended, but not required.

The frmsal.jar file is the "Forms Standalone Launcher" (FSAL). It is version specific; therefore it is not supported to use frmsal.jar from one installation against another. A checksum is also used to help ensure that the file is properly matched to the server and its Forms' version. This checksum is also helpful in ensuring that the launcher (frmsal.jar) has not been accidentally or maliciously replaced. As a result, you cannot use frmsal.jar downloaded from a Microsoft Windows server to run an application against a UNIX/Linux server or vice versa. The launcher is not client platform specific. It can be used on Microsoft Windows, Unix/Linux, or Apple Mac client machines. Essentially, any platform that supports running a Forms certified Oracle Java version can use FSAL.

All installations come with a Usage/Syntax web page that illustrates how to use the launcher, as well as providing a download link for the frmsal.jar file. The page can be accessed using a URL like this:

```
https://yourserver:port/forms/html/fsal.htm
```

Java

FSAL is a Java application and therefore Oracle Java is required on the user's machine. However, unlike running a Forms application embedded in a browser where a specific Java type (e.g. JRE – Plug-in) is required, FSAL supports using any Oracle Java distribution that supports running a Java application and was certified with the Oracle Forms version being used. Depending on the user's platform and the desired Oracle Java version to be used, there are several possible distribution types available. The following distributions are available for most common end-user platforms and can be used to run FSAL. In the Java 8 family; **JRE**, **JDK**, and **Server JRE** are available in both 32-bit and 64-bit distributions. Java 11 and newer are only available as a 64-bit **JDK** distribution, but can be used with FSAL. Depending on the application's needs, which distribution and/or version you choose may vary. It is recommended that the contents of the Oracle Java installation chosen be carefully reviewed and/or your application be thoroughly tested before moving to production. Here is an overview of each Java distribution.

Java Runtime Environment - JRE

Java 8 only. The JRE installs most components needed for typical end-users. It includes everything needed to run a local Java application, as well as the Java Deployment components (i.e. Java Web Start and Java Plug-in). Java Deployment components are not available in major releases newer than Java 8.

Java Development Kit - JDK

The JDK, available in all versions, is generally for Java developers and includes far more than a typical user should need. The JDK includes a complete JRE, the Java Deployment components, and tools for developing, debugging, and monitoring and maintaining Java applications.

Server JRE

Java 8 only. The Server JRE is primarily for deploying Java applications on servers. It includes tools for JVM monitoring and tools commonly required for server applications, but does not include the Java Deployment components needed for browser integration, for example Java Plug-in, Java Web Start, auto-update, and an installer are not included. This distribution is delivered as a zip file that must be manually extracted. Because this distribution does not perform a software installation and is much more lightweight than the JRE and JDK distributions, this option is ideal for the FSAL when used along with a customized startup script or similar.

Custom JRE

Beginning with Oracle Java version 11, it is now possible to create a custom JRE distribution. Explaining how to create such a distribution is beyond the scope of this document. However, in order to create a custom JRE the **`jdeps`** utility (used to determine which Java modules are needed) and **`jlink`** utility (used to create the custom JRE) will be needed. The **`jpackage`** utility can be used to package your custom JRE into an installable distribution if desired. Refer to the Java documentation for details on how to use these new utilities.

https://docs.oracle.com/en/java/javase/17/docs/specs/man/index.html

Be sure to check the Fusion Middleware Certification Guide to ensure that the Java version used is one that is certified for use with the Forms version used.

Running an Application with FSAL

The Basics

As mentioned earlier in this paper, FSAL is intended to allow users to run and interact with Forms applications by launching applications without the use of a web browser. Although it may be technically possible, this utility is not intended and not recommended to be used for non-interactive purposes (e.g. unattended load testing). Attempting to launch too many simultaneous or sequential sessions (e.g. more than a person would reasonably require) on the user's machine may result in unpredictable and undesirable behavior.

FSAL, like a browser, needs to know where to find the application to run. In a browser, hyperlinks are sometimes used to launch other web pages or even Forms applications. Because FSAL is fully removed from a browser, you cannot use a web page to launch Forms applications hosted by FSAL without some configuration changes being performed on the user's machine. Using custom protocol handlers (`fsal://` and `fsals://`), which is supported beginning with Forms 12.2.1.19, is one way in which FSAL can be launched from a web page. Knowing the complete Forms application URL is necessary in order to service the request. Currently, the use of URL redirects or rewrites is not supported, but may be technically possible depending on the server configuration. FSAL expects to receive a fully qualified URL that points to the Forms environment. A desktop shortcut or script/batch file can be used in place of a hyperlink in order to make starting the application simple and less error prone.

Running an application with FSAL is easy once the above requirements have been met. To start an application with FSAL, do the following:

1. Open a shell (e.g. DOS on Microsoft Windows) and verify the desired and certified Oracle Java version is found.

   ```
   java –version
   ```

   The result should indicate the desired Oracle Java version. If not, the system's PATH may not be properly set. Make the needed correction then continue.

2. Change directories to the user's home directory (assuming `frmsal.jar` was stored in the user's home directory) and execute the following command. Substitute the server name and port number in the example below. If the server is not running SSL enabled and instead is using HTTP, the protocol ("`http://`") can be omitted from the URL entry, as it will be assumed.

   ```
   java -jar frmsal.jar -url "https://server:port/forms/frmservlet?config=standaloneapp"
   ```

   In the example above, the application associated with the configuration section titled `[standaloneapp]` will run. You can use any configuration section as long as it contains the entries found in the `[standaloneapp]` example provided in the Forms Web Configuration.

3. Oracle Forms generated output typically seen in the Java Console will appear in the shell used to start the application. If the "`javaw`" or similar command is used rather than "java", a console may not be shown. If using the "`java`" command, closing the shell that started the application will terminate the application. Therefore it should remain open through the life of the session. This behavior can be altered to accommodate the application needs using various shell commands and associated switches. Refer to the operating system documentation for information on using the command shell on the user's platform.

The basic details above can be used to create a desktop shortcut, batch script file, or custom executable that can be used to make launching the application easier and more seamless.

Refer to the FSAL References section of this document for a complete list of command line arguments. This information can also be found on the FSAL Usage page provided in the installation. To access the Usage page, enter this URL in a browser:

```
http://server:port/forms/html/fsal.htm
```

File Caching

Similar to the Java Plugin or a browser, FSAL attempts to cache (store locally) files that may be reused the next time an application is launched. This helps to improve startup performance. When starting FSAL from a Unix/Linux shell or Windows DOS shell, the directory where cached files will be stored is displayed in the shell output during the loading process. Also displayed is whether the files are being downloaded from the server or reused from the existing cache. In the case where files are downloaded from the server, text similar to the following will be shown:

```
Inspecting archive files in cache directory
C:\Users\jdoe\AppData\Local\Temp\frmsal\<server name>\12.2.1.19

Downloading archive file frmall.jar to cache subdirectory 8ymuqdqvdfe13a0d5vvema0dh
```

In the case where the files previously cached will be used, text similar to the following will be shown:

```
Inspecting archive files in cache directory
C:\Users\jdoe\AppData\Local\Temp\frmsal\<server name>\12.2.1.19

Using cached archive file frmall.jar from cache subdirectory 8ymuqdqvdfe13a0d5vvema0dh
```

To control the location of the cache files, change the location of `tmpdir` when starting the application. Consider the following example for Microsoft Windows users. Note that it uses the value of the Windows system variable %USERNAME% as the first level directory.

```
java -Djava.io.tmpdir=C:/%USERNAME%/fsal -jar frmsal.jar -url
“https://server:port/forms/frmservlet?config=standaloneapp”
```

When the user is on a shared Unix/Linux platform this may be helpful because the downloaded cached files will be owned by the first user who ran the application. As a result, subsequent users may not have sufficient permissions to overwrite the old cached files with new ones. By creating unique locations for each user this problem can be avoided.

```
java -Djava.io.tmpdir=/u01/$user/fsal -jar frmsal.jar -url
"https://server:port/forms/frmservlet?config=standaloneapp"
```

For some applications, it may not be desirable to use cached files, but instead always download from the server. This may also be true when troubleshooting technical issues. To disable file caching, use Fusion Middleware Control (FMC) to edit the FSAL template file basesaa.txt or webutilsaa.txt, based on your application. Add the following line:

```
ignoreSaaCache=%ignoreSaaCache%
```

In the Forms Web Configuration (formsweb.cfg), add a new parameter **`ignoreSaaCache`** to the **`[standaloneapp]`** configuration or whichever configuration is being used to run the application. Set its value to **`TRUE`**. Cached files will now be ignored and downloading files from the server will occur each time the application is started.

Running Audio Enabled Applications with Java 11+

Oracle Java 11 or newer "Long Term Support" (LTS) releases can be used to run FSAL. However, Java 11 and newer no longer includes JavaFX. JavaFX is necessary for the audio feature in Forms to function properly. If the applications do not use the Forms audio feature, these steps can be ignored. To enable JavaFX the following steps would be necessary on the user's machine.

1. On the user's machine, download and extract the JavaFX SDK from *Gluon* (not affiliated with Oracle) into a directory where the user has runtime (read, execute) permissions. Download the version that most closely matches the Java version used to run FSAL. JavaFX is now an Open Source project. However, be sure to review the Terms of Use and support options before downloading and using.

   https://gluonhq.com/products/javafx

   Note that JavaFX versions newer than 8 are not supported by Oracle Support. Versions newer than 8 are supported and licensed by Gluon and/or the open source community found here:

   https://openjfx.io

   https://github.com/openjdk/jfx

2. Alter the typical FSAL startup command to include JFX. For example:

   ```
   java --module-path C:/javafx-sdk-17.0.4/lib --add-modules=javafx.media,javafx.swing -jar
   frmsal.jar -url "https://server:port/forms/frmservlet?config=standaloneapp"
   ```

Using Java Script Integration with FSAL Applications

Although the Forms Standalone Launcher and the applications it runs are not related to a web browser or its applications, it is possible to communicate with a web page from an FSAL application. For details, refer to the Forms 12c New Features Guide or the Working with Oracle Forms guide for more information.

Running Through a Proxy Server

In many cases, users will access a Forms application while within a corporate network. In some cases, this means that the user's machine requires the appropriate proxy configuration in order to access both internal and external content. In the case of using FSAL, the browser and system level settings may not be visible to the shell that launches the application. Therefore, it may be necessary to include such settings at the time FSAL is run. Because FSAL is a Java application, it is Java (e.g. java.exe) that must be aware of the needed proxy settings. There are several ways in which to inform Java of these settings.

To use the system proxy settings, do the following:

```
java -Djava.net.useSystemProxies=true –jar frmsal.jar -url
"https://server:port/forms/frmservlet?config=standaloneapp"
```

The use of this Java option (`-Djava.net.useSystemProxies=true`) will cause Java to attempt the call using the proxy settings provided at the system level. It should be noted that this method may not be supported when the system is configured to use Automatic Configuration Scripts (e.g wpad.dat). Refer to the official Java documentation for details.

Alternatively, proxy settings can be specifically included. Here are two examples:

```
java -Dhttps.proxyHost=<proxyserver> -Dhttps.proxyPort=<proxyserver port number> -
Dhttps.nonProxyHosts=”localhost|example.com” –jar frmsal.jar -url
“https://server:port/forms/frmservlet?config=standaloneapp”
```

```
java -Dhttp.proxyHost=<proxyserver> -Dhttp.proxyPort=<proxyserver port number> -
Dhttp.nonProxyHosts=”localhost|example.com” –jar frmsal.jar -url
http://server:port/forms/frmservlet?config=standaloneapp
```

Reference: https://docs.oracle.com/javase/8/docs/technotes/guides/net/proxies.html

Launching FSAL with a Custom Protocol Handler

Custom protocols have been used in web browsers nearly since the beginning of their existence. Typically, protocols like `http`, `https`, `ftp`, and even `mailto` are what you might expect to use or see when entering an address in a browser's address bar. However, it is also not uncommon to see special protocols like `zoommtg` to open a Zoom meeting or `slack` to open the Slack client app to a particular user or channel or message.

Beginning with Forms 12.2.1.19 doing similar with Forms Standalone Launcher (FSAL) is also possible. FSAL has been enhanced to recognize two special protocols; **`fsal`** for non-SSL requests and **`fsals`** for SSL requests. Using a custom protocol handler, a hyperlink from a web page can be used to launch a Forms app using FSAL. However, in order to achieve this the custom protocol must be registered on the user's machine and the desired web page would need to include a properly formatted hyperlink.

To create the hyperlink in the web page, simply create a hyperlink in the same manner used for any other. Adding HTML like the following will introduce a hyperlink that can be used to launch FSAL with a named Forms configuration (standaloneapp). Of course, providing an appropriate server and port is necessary too.

```
<a href=”fsal://<SERVER>:<PORT>/forms/frmservlet?config=standaloneapp”>Run FSAL</a>
```

The steps needed to register this new protocol on the user's machine will vary depending on the operating system used by the user. Several approaches may be possible regardless of the operating system in use. Below is just one example with the assumption the user's machine is running Microsoft Windows.

The following assume that:

- The latest Forms frmsal.jar is stored in the user's home directory.
- The latest Oracle Java 8 (32-bit) JRE is installed.

The following commands can either be run individually on a Windows DOS command prompt or executed from a script. Regardless of the approach, the user must have Administrator privileges since this will write to the Windows Registry.

Warning:

Manipulating the Windows Registry is risky and can cause un-reversible damage if performed incorrectly. Be sure to create a Registry backup and System Restore Point before attempting. Do not attempt this if you are unfamiliar with editing the Windows Registry.

In the following commands, be sure to carefully review the last entry in each section. Be sure the path to “javaw.exe” and the path to “frmsal.jar” (on the user's machine) represents your situation. Note that the Java path used is a special path created when installing the JRE only. If using the JDK, this path will need to be updated.

Commands for non-SSL Requests

```
reg add HKEY_CLASSES_ROOT\fsal /t REG_SZ /d "Oracle Forms Standalone Launcher" /f

reg add HKEY_CLASSES_ROOT\fsal /v "URL Protocol" /t REG_SZ /d "" /f

reg add HKEY_CLASSES_ROOT\fsal /v "UseOriginalUrlEncoding" /t REG_DWORD /d "00000001"

reg add HKEY_CLASSES_ROOT\fsal\shell /f

reg add HKEY_CLASSES_ROOT\fsal\shell\open /f

reg add HKEY_CLASSES_ROOT\fsal\shell\open\command /t REG_SZ /d "\"C:\Program Files
(x86)\Common Files\Oracle\Java\javapath\javaw.exe\" -jar %USERPROFILE%\frmsal.jar -url
\"%1\"" /f
```

Commands for SSL Requests

```
reg add HKEY_CLASSES_ROOT\fsals /t REG_SZ /d "Oracle Forms Standalone Launcher with
SSL/TLS" /f

reg add HKEY_CLASSES_ROOT\fsals /v "URL Protocol" /t REG_SZ /d "" /f

reg add HKEY_CLASSES_ROOT\fsal /v "UseOriginalUrlEncoding" /t REG_DWORD /d "00000001"

reg add HKEY_CLASSES_ROOT\fsals\shell /f

reg add HKEY_CLASSES_ROOT\fsals\shell\open /f

reg add HKEY_CLASSES_ROOT\fsals\shell\open\command /t REG_SZ /d "\"C:\Program Files
(x86)\Common Files\Oracle\Java\javapath\javaw.exe\" -jar %USERPROFILE%\frmsal.jar -url
\"%1\"" /f
```

Once the above have been successfully executed, the protocols `fsal://` or `fsals://` can be used in a browser in place of `http://` or `https://` `For registering custom protocol handlers on other operating systems, refer to the Operating System documentation or contact the vendor for assistance.`

Security Tips

Ensuring the highest degree of security for an application, the data exchanged within it, and the network on which it is hosted should be considered the most important aspects of any application deployment. A weakened security layer can be responsible for sensitive data breaches and malicious system attacks. As an application developer, administrator, or any other role in IT, it is your responsibility to ensure that proper security efforts are being used to protect the applications, the data, and their hosting systems.

This section will offer several ways that can be used to help improve the security related to using the FSAL. It is important that you use these suggestions only as examples. It is your responsibility to understand the concepts and research any that you do not fully understand. Improperly implementing any security configuration may put your system at risk. So, carefully review and test your changes before assuming they are correct. The suggestions provided here may not be unique to FSAL or even Oracle Forms in general. Many are standard suggestions that relate to the technologies in use. Therefore, the availability of additional information is widespread. Do not use this paper as the only source of information for securing your application, its data, and its environment.

Be sure to also review the Resources section included at the end of this document.

Secure Socket Layer

In this document, the terms “KeyStore” and “TrustStore” may be used interchangeably.

Secure Socket Layer (SSL) and Transport Layer Security (TLS) are cryptographic protocols used to provide encrypted communication between a source and destination of network traffic. These protocols works by creating a trusted connection, which is most often established by a public and private key exchange. The details of how SSL/TLS works are beyond the scope of this document.

Using SSL/TLS to run any application communicating on the network is very important with regard to securing data. The use of SSL/TLS for running any and all applications should be considered a requirement and not optional. To run an application using SSL/TLS with FSAL requires that the SSL/TLS certificate public key be included in the user's Java TrustStore. This may require the certificate(s) be manually imported if that has not already been included in the TrustStore or provided by a known Certificate Authority (e.g. DigiCert, Entrust, Comodo, etc). Specifically, this key must be imported into the Java TrustStore that is being used by FSAL to run the form. Although often only the "root" certificate will be needed, in some cases, more than one certificate may need to be imported, as a chain of certificates may exist (e.g. root, intermediate, user). If you attempt to run an application using SSL/TLS and a needed certificate is not found or was improperly imported, a Java error similar to one of the following, likely may be presented:

```
java.security.cert.CertificateException: No subject alternative names present …
```

OR

```
java.security.cert.CertificateException: No name matching <server:port> found …
```

OR

```
javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: …
```

Following one of the above or similar errors will be an FSAL error. This is because a secure connect could not be established and the parameters needed to start and run the application were not permitted to be downloaded.

```
FRM-92490: Unable to fetch applet parameters from server.
```

Configuring SSL/TLS with FSAL

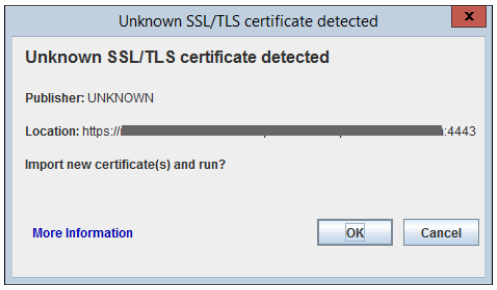
Beginning with Forms 12.2.1.19, FSAL provides two options for importing and storing SSL/TLS certificates.

The default configuration will be to use the FSAL Certificate Importer. This dialog will be presented to users when the certificate delivered by the server is not recognized by the user's system. Upon the user accepting the certificate and granting permission to import it, the new certificate will be inserted into a newly created Java TrustStore used by Forms/FSAL. This FSAL created TrustStore will be used by FSAL for all subsequent requests unless disabled by the user by setting the runtime switch `cert_truststore` to "`java`".

The alternative option for performing the configuration will be to use the Java provided tools to manually import the missing certificate(s). This is the same option that was available in previous versions. Refer to the command line options provided near the end of this document for details on reverting to the previous behavior.

Using the FSAL Certificate Importer

Using the FSAL Certificate Importer is mostly automatic. FSAL is used as described previously in this document. Upon receiving initial feedback from the server, FSAL will check the TrustStore and determine if the delivered certificate is known. If known and deemed trusted, the application will run without interruption. If the certificate sent by the server is unknown, the user will be presented with a dialog indicating its findings and allowing the user to decide how to proceed (i.e. import the certificate or cancel the operation). The user will simply click on Ok to continue or Cancel to end the operation without importing. If desirable, details about the delivered certificate can be reviewed before continuing by clicking on the "More information" link found on the dialog.



Unknown SSL/TLS certificate detected

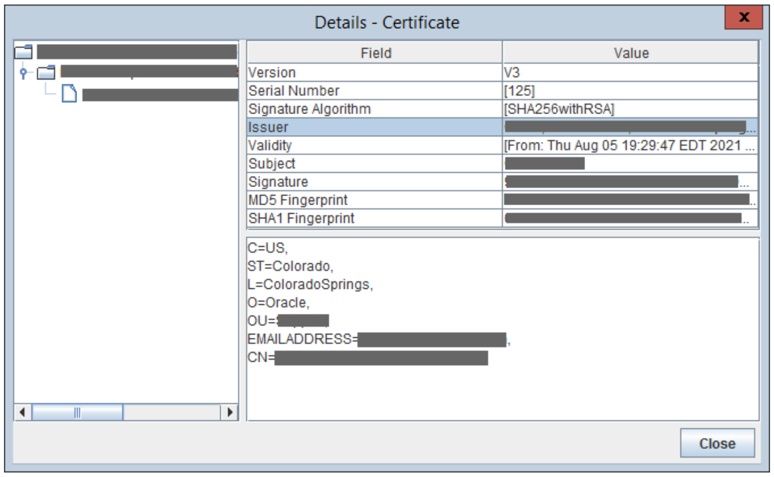
Unknown SSL/TLS certificate detected

Publisher: UNKNOWN

Location: https://[redacted]:4443

Import new certificate(s) and run?

Figure 1 - FSAL Certificate Importer example.



Details - Certificate

[redacted]

Field	Value
Version	V3
Serial Number	[125]
Signature Algorithm	[SHA256withRSA]
Issuer	[redacted]
Validity	[From: Thu Aug 05 19:29:47 EDT 2021 ...
Subject	[redacted]
Signature	[redacted]
MD5 Fingerprint	[redacted]
SHA1 Fingerprint	[redacted]

C=US,
ST=Colorado,
L=ColoradoSprings,
O=Oracle,
OU=[redacted]
EMAILADDRESS=[redacted],
CN=[redacted]

Figure 2 - FSAL Certificate Importer Details Viewer example.

Permitting the certificate to be imported and allowing the application to run will result in a custom Java TrustStore being created/updated in the user's home directory, specifically in:

```
C:\Users\<USER NAME>\Oracle\forms
```

The TrustStore file name will be `formstruststore`. Changing the name of the generated TrustStore is not supported. If the file is not found, a new empty file will be created. Because this is a Java generated and standard TrustStore file, the tools found in the Oracle Java installation (e.g. keytool.exe) can be used to administer this file if necessary.

An option to automatically import unknown certificates without user intervention is possible using the `autoImportCert` switch. This option is not recommended and is provided mostly for testing purposes. Here is an example of how to use it:

```
java -jar frmsal.jar -url "https://server:port/forms/frmservlet?config=standaloneapp" -
autoImportCert true
```

For additional information, refer to the FSAL command line arguments listed on the Usage web page mentioned earlier in this document. The list of command line arguments is also included near the end of this guide.

Configuring the KeyStore Manually

If using the FSAL SSL/TLS Certificate Importer is not desirable, follow these steps prior to using FSAL. It may also be necessary to complete these steps if a compatibility issue is identified with the certificate(s) in use and the FSAL Certificate Importer. Some non-standard or self-generated certificates may not be compatible with the FSAL Certificate Importer.

1. Obtain an SSL/TLS certificate from a known Certificate Authority and configure Oracle HTTP Server (or WebLogic Server) according to the instructions provided by the documentation for those components.

2. The public key portion of the certificate is be needed on the user's machine. If you do not have the public key, there are several ways to obtain it. Obtaining the key chain can be done from any machine that has access to the server. Here is an example of how to obtain the key(s) using the `openssl` command. This is pre-installed on most Unix/Linux platforms, but can be obtained for Microsoft Windows.

   ```
   openssl s_client -showcerts -connect server:port > output.txt
   ```

   In the above example, replace **`server:port`** with the server name and SSL/TLS port number. The result of running this command will be saved in a file named "output.txt". The output will contain one or more certificates (depending on the certificate type and how it was created). The certificate is the contents between and including the BEGIN and END header/footer. Copy each certificate to its own text file. Be sure to only include the BEGIN and END text exactly as shown below and the contents between them. Do not include any extra lines above or below the **`-----BEGIN CERTIFICATE-----`** or **`-----END CERTIFICATE-----`** when saving the file, but do include these header/footer entries.

   Here is an example:

   ```
   -----BEGIN CERTIFICATE-----

   MIIFKDCCBBCgAwIBAgIBPTANBgkqhkiG9w0BAQsFADCBrjEpMCcGA1UEAxMgTmlj

   Y2subWFuc0BvcmFjbGUuY29tMRAwDgYDVQQLEwdTdXBwb3J0MQ8wDQYDVQQKEwZP

   cmFjbGUxGTAXBgNVBAcTEENvbG9yYWRvIFNwcmluZ3MxETAPBgNVBAgTCENvbG9y

   . . .

   -----END CERTIFICATE-----
   ```

   Alternatively, run the following to directly generate the needed certificate file.

   ```
   openssl s_client -showcerts -servername <server name> -connect <server:port> | openssl
   x509 -outform PEM > cert.cer
   ```

3. Import the public key(s) into the KeyStore on the user's machine (e.g. cacerts). If it is a certificate chain, be sure to import all keys in the chain (etc. signer, intermediate, root, etc). This means it may be necessary to run the `keytool` utility more than one time. To import the certificate public key(s) use the Java `keytool` utility, which is included in most Oracle Java distributions.

   Here is an example of how import each certificate using the "keytool" utility. Refer to the Oracle Java documentation for details on how to use the keytool utility.

   ```
   keytool -import -alias <server_name> -TrustStore <JAVA_HOME>/jre/lib/security/cacerts -
   file cert.cer
   ```

   The "alias" must be unique, meaning if there are more than one certificate in the chain each must have a unique alias. However the root should be imported first and should use the server name as its alias. Attempting to use an alias that was previously used will result in an error. It is recommended that the server name be used in the alias. For example the root certificate should be imported first and should be named `myserver`, the second `myserver2`, and the third `myserver3`.

4. Run FSAL with SSL, as shown in the example provided in the previous section of this paper, "Running an Application with FSAL". However, be sure to include the switch `-cert_truststore` set to `java` in order to bypass the automatic certificate importer.

```
java -jar frmsal.jar -url "https://server:port/forms/frmservlet?config=standaloneapp" –
cert_truststore java
```

Troubleshooting SSL/TLS

TIP 1:

If certificates were manually inserted into the Java Keystore, verify the correct KeyStore was updated. By default, a KeyStore is found in `JRE_HOME/lib/security` or in the user's home directory. The default file name (when not using the FSAL Certificate Importer) is **`cacerts`**. Verify that the file modified date/time is the same as when the `import` command was executed. When using the FSAL Certificate Importer the TrustStore name is **`formstruststore`**.

TIP 2:

Verify that all certificates in the chain have been imported. In many cases, only the root certificate is needed. If it is believed that there is a certificate chain, all the certificates can be listed for review using the following Java command:

```
keytool -list -keystore "C:\java\jdk1.8.0_341\jre\lib\security\cacerts"
```

TIP 3:

If the KeyStore or TrustStore updated is not in the JRE_HOME or user home directory, copy it to the JRE_HOME. Be sure to create backup files of any files before overwriting.

TIP 4:

Enable Java SSL/TLS debugging. To enable debugging mode, run FSAL and the desired form as follows:

```
java -Djavax.net.debug=all –jar frmsal.jar -url
https://yourserver:port/forms/frmservlet?config=standaloneapp
```

The output can also be redirected to a text file as follows (assumes Microsoft Windows):

```
java -Djavax.net.debug=all –jar frmsal.jar -url
"https://yourserver:port/forms/frmservlet?config=standaloneapp" >
C:/existing_directory/output.txt
```

TIP 5:

If the KeyStore or TrustStore that contains the needed certificate(s) is not the default (in the default location), the "keystore" or "truststore" switch can be used to explicitly reference the desired file.

```
java -Djavax.net.ssl.keystore=C:/somewhere/mycacerts -jar frmsal.jar -url
"https://server:port/forms/frmservlet?config=standaloneapp"
```

or

```
java -Djavax.net.ssl.truststore=C:/somewhere/mycacerts -jar frmsal.jar -url
"https://server:port/forms/frmservlet?config=standaloneapp"
```

Signed Code

When an application is requested to be launched, it is helpful to know who owns the application. This helps to ensure that the application owner is known and trusted by the user. If not trusted, the user can choose to cancel the operation. This application owner identity is the result of including a digital signature (certificate) within the application. For Oracle Forms, a digital signature certificate is added to the Oracle provided Forms Java JAR files. Refer to the "`jarsigner`" utility and its documentation for details on how to add a certificate to your own JAR files.

Forms, Java Plug-in, and Java Web Start

To run a Forms application from a browser with the Java Plug-in or Java Web Start, the user likely clicks on a hyperlink or bookmark or manually enters a URL into a browser. Unrelated to the use of Oracle Forms, the simple fact that the user is using a browser puts their system at potential risk. With so many malicious web sites accessible on the Internet, stumbling upon a malicious site, intentionally or unintentionally is almost inevitable. Fortunately, the Java Plug-in and Java Web Start have special security features built-in to help mitigate the chances of malicious Java applications being run on the user's machine. One of these key security features is a requirement to only permit signed (and trusted) applications to be run. Unsigned applications are not permitted and are blocked. Since there is cost and organizational verification associated with obtaining a trusted code signing certificate (digital signature), attempts to deliver malicious content in this manner are fairly rare, but are possible.

All Oracle Forms provided Java JAR files that run on the user's machine are signed in order to comply with Java's security requirements mentioned above. This allows Forms applications to safely run in a browser or with Java Web Start. It is further required that custom JAR files (those not provided by Oracle) also be properly signed with a trusted certificate. Signing JARs not provided by Oracle is the customer's responsibility. Using self-generated certificates is not recommended and should be avoided. Although using self-generated certificates may seem to function, support for this functionality may be removed from Java in the future.

FSAL and Signed JARs

JAR signing is recommended for all deployments, even when using FSAL. Since the inherent risk associated with using a browser is mostly avoided when using FSAL, this client configuration is quickly becoming the preferred approach. That said, carefully considering all security aspects of application deployment remains important.

Although running native Java applications like FSAL do not have the same level of built-in security checks found in the Java Plug-in and Java Web Start, limited levels of signed code verification can be enabled.

The following has been ***deprecated*** in Java versions newer than version 11. Warnings as such may appear if enabling this functionality with Java 17 or newer. These instructions are provided for legacy purposes only.

To enable signed code awareness for FSAL and specifically your application, follow these steps:

1. Obtain the public keys associated with all JAR files used in the application. Oracle provided JARs are signed when delivered and this certificate is included in the KeyStore by default. If the signer's public key for the JAR files used is not available, use the following Java command for each of the JARs to obtain their certificates. The first certificate returned should be the signer certificate and the one needed for the steps that follow.

   ```
   keytool -printcert -rfc -jarfile yourJarFile.jar
   ```

2. On the user's machine, import each public key into the Java KeyStore that will be used at runtime.

   ```
   keytool -importcert -alias foo -file C:\foo.pem
   ```

   The above alias can be any alphanumeric string, however be sure to remember it as it will be needed in a coming step. Also, because the `–keystore` switch was not included, a KeyStore in the user's home directory will be used (created if one does not exist). The file will be named ".keystore". To use a different KeyStore and/or KeyStore name, add the `–keystore` switch as desired. Repeat the above step for each certificate.

Refer to the Resources section of this document for helpful links to using the “keytool” utility.

3. Create a .java.policy file in the user’s home directory if one does not exist or if you want a fully customized file. Otherwise, use the default file. <JAVA_HOME>\jre\lib\security\java.policy   Add the following to it:

```
keystore ".keystore";

grant signedBy "foo" {

   permission java.security.AllPermission;

};
```

   The “foo” reference is a pointer to the alias used in Step 2. Add additional “grant signedBy” sections for each key that was imported in Step 2. The value of `keystore` is a pointer to the KeyStore file you expect to use.

4. Run your Forms application using FSAL with this slightly modified command entry:

```
java -Djava.security.manager -jar frmsal.jar -url
"https://yourserver:port/forms/frmservlet?config=standaloneapp"
```

When running the application, JAR certificates will be matched with those in the KeyStore. If they do not match, the app will not run. In the above example, it is assumed that the default java.policy is used or a .java.policy was created in the user’s home directory. You can alternatively access a custom policy file from an alternative directory on the user’s machine or a remote location (URL).

```
java -Djava.security.manager -Djava.security.policy=${user.home}/foo.policy

-jar frmsal.jar -url "https://yourserver:port/forms/frmservlet?config=standaloneapp"


java -Djava.security.manager -Djava.security.policy=https://yourserver:port/foo.policy

-jar frmsal.jar -url "https://yourserver:port/forms/frmservlet?config=standaloneapp"
```

To ensure users are taking advantage of this validation, a startup script or similar should be used. Using a script will help prevent the possibility of typographical errors or the possibility of not using the security manager.

Single Sign-on

Single Sign-on is an authentication concept that offers many valuable advantages to web applications. The idea of integrating Forms with Single Sign-on (SSO) is often seen as having at least two significant benefits over using the Forms database login functionality. The first is that the user only has to be authenticated to an application one time while their browser remains open. This authentication can be shared with other browser applications or other Forms applications. This means the user would not be required to login to each application, but rather only the first. This applies to all applications protected by the same SSO server and applications running or launched from the same browser session. The second advantage is that the database credentials for running Forms applications would be hidden from the user.

To use SSO with FSAL you must perform the same configuration steps needed for using SSO with Forms in any other case. For information about how SSO is configured with Forms and how it works, refer to the “Working with Oracle Forms Guide”. Once this common configuration has been completed, add **`ssoMode=%ssoMode%`** to the Forms template files basesaa.txt and webutilsaa.txt, using Fusion Middleware Control. To enable SSO for a specific application, set **`ssoMode=true`** in the desired application configuration of the Forms Web Configuration settings associated with the application to be protected.

FSAL References

This section of the guide includes web configuration parameters, command line arguments, and several related error messages. For complete details, refer to the "Working with Oracle Forms Guide".

FSAL Web Configuration Parameters

For additional information about Forms Web Configuration parameters, refer to "Working with Oracle Forms".

SERVLET/APPLET PARAMETER	DESCRIPTION
ignoreSaaCache	Specifies whether to ignore existing cache and download fresh file(s) from the server. Valid values: TRUE, FALSE Default: FALSE
ignoreMissingSaaArchives	Specifies whether to ignore configured but missing JAR files at startup. This should generally only be used for testing and debugging. When set to TRUE, any occurrence of a missing archive will result in an error but the session will attempt to continue. This is the default behavior. When set to FALSE, FSAL will attempt to continue if possible, but fail if not. Valid values: TRUE, FALSE Default: FALSE
fsalJavaVersion	Specifies the required Java version to run the application. Refer to the Web Configuration Parameters table near the end of the Working with Oracle Forms Guide for possible value combinations.
ssoSaaBrowserLaunchTimeout	Specifies, in seconds, how long the Forms servlet will wait for the initial request from the browser that was launched by FSAL for SSO authentication. If the interval expires, the fatal error FRM-93249 will be reported. Valid values: Integers in the range 1-300. Default: 15
ssoSaaBrowserPageTimeout	Specifies, in seconds, how long the Forms servlet will wait for the user to enter data into a browser page during SSO authentication for an FSAL application. If the interval expires, the fatal error FRM-93382 or FRM-93383 will be reported. Valid values: Integer >= 15, or 0 (wait indefinitely). Default: 0
ssoSaaWaitInterval	Specified the interval, in seconds, at which FSAL reissues requests to the Forms servlet while SSO authentication is proceeding in the launched browser. Larger values reduce network traffic but increase the chances of a timing out (thereby producing the fatal error FRM-93248). Valid values: Integer >= 5, or 0 (do not reissue requests). Default: 25
ssoSuccessLogonURL	The URL to redirect to if SSO authentication completes successfully for an FSAL application.

FSAL Command Line Arguments

For additional information about FSAL command line arguments, refer to "Working with Oracle Forms".

COMMAND LINE ARGUMENT	DESCRIPTION
-url	Specifies the fully qualified URL needed to run the Forms application. The URL should include the necessary configuration reference that includes the needed FSAL settings (e.g. config=standaloneapp). The URL must be surrounded by quotation marks. When not using SSL/TLS, the protocol http:// can be omitted from the URL entry.
-t	Specifies, in milliseconds, the amount of time the launcher should wait for the server to provide its initial response before timing out. Valid values: Positive integer >= 1 Default: 60000 (ms)
-showConfig	Specifies if the Forms web configuration parameters should be display on command line when the application is loaded. Valid values: TRUE, FALSE Default: FALSE
-showDetails	Specifies whether to display additional details on application loading and startup. Valid values: 0, 1, 2, 99 Value descriptions: 0 No additional information should be displayed. 1 Show location from where resources are loaded. 2 Show SSL/TLS related information when certificate information is not in the FSAL TrustStore. 99 Shows all the details. Other values are reserved for future use. Default: 0
-changeFSALStorePass	Specifies whether to change the password of FSAL custom TrustStore. A default password is set at creation time. That password is: `changeit` In most cases, there is little reason to set a password for the truststore on a user's machine unless it contains private key/certificate information. Valid values: TRUE, FALSE Default: FALSE
-autoImportCert	Specifies whether the SSL/TLS certificates should be imported without any user interaction. This argument is ignored if `cert_truststore` is set. Valid values: TRUE, FALSE Default: FALSE
-cert_truststore	Specifies whether to use the Forms TrustStore or a custom KeyStore for storing SSL/TLS certificates. A value of "`forms`" or unset will result in using a Forms generated Java TrustStore. A value of "`java`" will cause FSAL to look in the Java default KeyStore for storing and verifying SSL/TLS

	certificates. When using the Java default store, you must manually import needed certificates into it before running any SSL/TLS application. Valid values: forms, java Default: forms
-clearcache	Specifies whether to clear all FSAL cache. The `-url` argument will be ignored if used with `-clearcache`. Valid values: TRUE, FALSE Default: FALSE

Resources

https://www.oracle.com/application-development/technologies/forms/forms.html

https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html

https://docs.oracle.com/javase/tutorial/security/tour2/step2.html

https://docs.oracle.com/javase/tutorial/security/toolsign/rstep1.html

https://docs.oracle.com/javase/tutorial/security/toolsign/rstep2.html

https://docs.oracle.com/javase/tutorial/security/toolsign/rstep3.html

https://docs.oracle.com/javase/8/docs/technotes/guides/security/PolicyFiles.html

https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html

https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/ReadDebug.html

https://openjdk.java.net/jeps/411

https://docs.oracle.com/javase/8/docs/technotes/guides/net/proxies.html

https://en.wikipedia.org/wiki/Transport_Layer_Security

https://www.digicert.com/what-is-an-ssl-certificate

https://en.wikipedia.org/wiki/Chain_of_trust

https://www.openssl.org

https://www.geeksforgeeks.org/difference-between-truststore-and-keystore-in-java

Connect with us

Call **+1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle