Platform SDK Developer's Guide

Use Cases

4/22/2025

Contents

- 1 Use Cases
  - 1.1 Introduction
  - 1.2 Genesys Server Use Cases
  - 1.3 Genesys Application Use Cases

Use Cases

Introduction

This page examines TLS functionality as a series of common use cases. Use cases are broken into two categories: server or application.

Examples and explanations are provided for some use cases, while others simply provide links to the related TLS documentation needed to understand the functionality.

Genesys Server Use Cases

Opening a TLS Port

Code snippets explaining how to open a basic TLS port are provided both with, and without using the Application Template Application Block:

- Opening a TLS port using the Platform SDK Commons Library
- Opening a TLS port using the Application Template Application Block

Opening a Mutual TLS Port (With Expiration, Revocation and CA Checks)

This use case is an advanced variation on opening a simple TLS port. As such, it already has a CA and expiration check, but needs additional parameters to turn on mutual mode and to enable a CRL check.

Mutual Mode

If TLS is configured programmatically, then the *mutualTLS* parameter should be set to *true* when creating an *SSLExtendedOptions* object:

```
SSLExtendedOptions sslOptions = new SSLExtendedOptions(true, (String) null);
```

If TLS is configured in Configuration Manager, then the *tls-mutual* parameter for the server port, application or host should be set to *1*. Please refer to the list of TLS parameters for details.

Revocation Check

If TLS is configured programmatically, then a valid path to the CRL file should be provided in the *crlFilePath* parameter when creating a trust manager:

```
X509TrustManager tm = TrustManagerHelper.createPEMTrustManager(
        "c:/cert/ca-cert.pem","c:/cert/crl.pem", null);
```

If TLS is configured in Configuration Manager, then the *tls-crl* parameter for the server port, application or host should contain the path to the CRL file located on server. Please refer to the list of TLS parameters for details.

Opening a FIPS-Compliant Port

FIPS mode is not a property of a port or application; it is defined mostly by the type of security provider in use and the OS/environment settings. For Java, the PKCS#11 security provider should be used to support FIPS; for .Net, FIPS is configured at the OS level (http://technet.microsoft.com/en-us/library/cc750357.aspx).

If TLS is configured programmatically, then a PKCS11 key/trust managers should be used:

```
X509TrustManager tm = TrustManagerHelper.createPKCS11TrustManager(
        new DummyPasswordCallbackHandler(), (String) null);
X509ExtendedKeyManager km = KeyManagerHelper.createPKCS11KeyManager(
        new DummyPasswordCallbackHandler());
```

If TLS is configured in Configuration Manager, then the *fips140-enabled* parameter for the server port, application or host should be set to "1". Please refer to the list of TLS parameters for details.

Note: This parameter is used to detect the security provider type to use. If this setting conflicts with other TLS parameters or points to a FIPS security provider that is not installed on host, then Platform SDK will generate an exception when attempting to accept or open a connection.

Genesys Application Use Cases

Opening a TLS Connection to a TLS Autodetect Server Port

TLS autodetect ports (also called upgrade mode ports) allow you to establish an unsecured connection to the server before specifying TLS settings. For details, please refer to Connecting to Upgrade Mode Ports in the quick start instructions.

Opening a TLS Connection to a Backend Server (With Expiration, Revocation and CA Checks)

Code snippets explaining how to open a basic TLS connection to a backend server are provided both with, and without using the Application Template Application Block:

- Configuring TLS for Client Connections using the Platform SDK Commons Library
- Configuring TLS for Client Connections using the Application Template Application Block

Opening a FIPS-Compliant Connection to a FIPS-Compliant Port

In this use case, the application does not need to provide any special behavior because the server will only handshake for FIPS-compliant ciphers. Details about setting up a FIPS-compliant port are described above.

Ensuring the Certificate is Checked with CA

If TLS is configured programmatically, then a valid CA certificate data should be provided when creating the trust manager:

```
X509TrustManager tm = TrustManagerHelper.createPEMTrustManager(
        "c:/cert/ca-cert.pem","c:/cert/crl.pem", null);
```

If TLS is configured in Configuration Manager, then the *trusted-ca* parameter for the port, connection, application or host should contain valid CA certificate data. Please refer to the list of TLS parameters for details.

Note: CA certificates are configured differently for each type of security provider. Please refer to the page on using and configuring security providers for detailed information.

Ensuring the Certificate Expiration is Checked

Certificate expiration is checked by default during the certificate validation process.

Note: If a server certificate is placed in a trusted certificates store on the client host, it will be automatically trusted without any validation. A trust certificates store should not include application certificates; instead, it should contain only CA certificates.

Handling a Certificate Revocation List

If TLS is configured programmatically, then a valid path to a CRL file should be provided in the *crlFilePath* parameter when creating trust manager:

```
X509TrustManager tm = TrustManagerHelper.createPEMTrustManager(
        "c:/cert/ca-cert.pem","c:/cert/crl.pem", null);
```

If TLS is configured in Configuration Manager, then the *tls-crl* parameter for the application connection, application or host should contain the path to the CRL file located on the application's host. Please refer to the list of TLS parameters for details.

Handling a User-Specified Cipher List

If TLS is configured programmatically, then the *enabledCipherSuites* constructor parameter should contain a list of allowed ciphers when the *SSLExtendedOptions* object is being created:

```
SSLExtendedOptions sslOptions = new SSLExtendedOptions(
        true, "TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA " +
        "TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA");
```

If TLS is configured in Configuration Manager, then the *cipher-list* parameter for the port, connection, application or host should be set to contain list of allowed ciphers. Please refer to the list of TLS parameters for details.