Platform SDK Developer's Guide

Quick Start

5/2/2025

Contents

Quick Start

Understanding Port Modes

TLS is configured differently depending on target port mode:

- default - Default mode ports do not use or understand TLS protocol.
- upgrade - Upgrade mode ports allow unsecured connections to be made, switching to TLS mode only after TLS settings are retrieved from Configuration Server.
- secure - Secure mode ports require TLS to be started immediately, before sending any requests to server.

Connecting to Default Mode Ports

Default mode is supported for all protocols; no specific configuration is needed for it to work.

Example:

```
Endpoint cfgServerEndpoint = new Endpoint(appName, cfgHost, cfgPort);
ConfServerProtocol protocol = new ConfServerProtocol(cfgServerEndpoint);
protocol.setClientName(appName);
protocol.setClientApplicationType(appType);
protocol.setUserName(username);
protocol.setUserPassword(password);
protocol.open();
```

It is also OK to specify explicit null parameters for the connection configuration and TLS parameters:

```
// Explicit null ConnectionConfiguration
Endpoint cfgServerEndpoint = new Endpoint(appName, cfgHost, cfgPort, null);

// Explicit null ConnectionConfiguration and TLS parameters
Endpoint cfgServerEndpoint = new Endpoint(appName, cfgHost, cfgPort, null, false, null, null);
```

Connecting to Upgrade Mode Ports

TLS upgrade mode is supported only for Configuration Protocol, since the TLS settings for connecting clients must be retrieved from Configuration Server. No specific options are required; the TLS upgrade logic works by default.

If a user has provided custom settings, then those settings are used if the TLS parameters received from Configuration Server are empty. The only requirement that the *tlsEnabled* parameter in the Endpoint constructor is **not** to true, otherwise the client side starts TLS immediately and the connection would fail because an upgrade mode port expects the connection to be unsecured initially.

```
// Setting tlsEnabled to true would cause failure when connecting to upgrade port:
Endpoint cfgServerEndpoint = new Endpoint(appName, cfgHost, cfgPort,
```

```
    connConf, true, sslContext, sslOptions);
```

Connecting to Secure Mode Port

Secure mode is supported for all protocols. TLS configuration objects/properties must be specified before the connection is opened, and the *tlsEnabled* parameter must be set to true. Secure port mode expects the client to start TLS negotiation immediately after connecting, otherwise the connection fails.

Example:

```
boolean tlsEnabled = true;
// Here, the minimal TLS configuration is used, see the following section for details
TrustManager trustManager = TrustManagerHelper.createTrustEveryoneTrustManager();
KeyManager keyManager = KeyManagerHelper.createEmptyKeyManager();
SSLContext sslContext = SSLContextHelper.createSSLContext(keyManager, trustManager);
ConnectionConfiguration connConf = new KeyValueConfiguration(new KeyValueCollection());
Endpoint cfgServerEndpoint = new Endpoint(appName, cfgHost, cfgPort,
    connConf, tlsEnabled, sslContext, sslOptions);
ConfServerProtocol protocol = new ConfServerProtocol(cfgServerEndpoint);
protocol.setClientName(appName);
protocol.setClientApplicationType(appType);
protocol.setUserName(username);
protocol.setUserPassword(password);
protocol.open();
```

TLS Minimal Configuration

Frequently, there is a need to quickly set up code for working TLS connections, dealing with detailed TLS configuration later. The minimal configuration settings described below do exactly that.

Platform SDK for Java

The following code creates an *SSLContext* object that can be used to configure a connection to a secure port or to configure a secure server socket. This code uses *EmptyKeyManager* which indicates that the party opening connection/socket would not have any certificate to authenticate itself, and *TrustEveryoneTrustManager* which trusts any certificate presented by the other party - even expired or revoked certificates.

```
boolean tlsEnabled = true;
TrustManager trustManager = TrustManagerHelper.createTrustEveryoneTrustManager();
KeyManager keyManager = KeyManagerHelper.createEmptyKeyManager();
SSLContext sslContext = SSLContextHelper.createSSLContext(keyManager,
trustManager);
```

Note: Connections using this configuration would have a working encryption layer, but they are not secure because they can neither authenticate themselves nor validate credentials provided by the other party.

Note: If a server uses mutual TLS mode, then it requires the client to present a certificate. Minimal configuration does not have certificates, so in this case the TLS negotiation would fail.

Platform SDK for .Net

Platform SDK for .Net requires less configuration, because it always uses the MSCAPI security provider and Windows Certificate Services (WCS) by default. The following code would trust all certificates located in the WCS Trusted Root Certificates folder for the current user account.

```
KeyValueConfiguration config = new KeyValueConfiguration(new KeyValueCollection());
config.TLSEnabled = true;
Endpoint ep = new Endpoint(AppName, Host, Port, config);
```

Note: If a server uses mutual TLS mode, then it requires clients to present a certificate. Minimal configuration does not have certificates, so in this case the TLS negotiation would fail.