Platform SDK Developer's Guide

Using and Configuring TLS Protocol Versions

12/16/2025

Using and Configuring TLS Protocol Versions

Java

The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications. Since there are various versions of TLS (1.0, 1.1, 1.2, and any future versions) and SSL (2.0 and 3.0), there needs to be a way to negotiate which specific protocol version to use. The TLS protocol provides a built-in mechanism for version negotiation so as not to bother other protocol components with the complexities of version selection.

> **Tip**
> By default, the following client-side TLS Protocol versions are enabled in Java:
>
> - Java 6, Java 7 – TLSv1
> - Java 8, Java 11 – TLSv1.2

Platform SDK for Java includes extended TLS support so that you can specify which version of TLS to use. There are several ways to accomplish this, as described below.

1) Using the `ConnectionConfiguration` class:

```
KeyValueConfiguration config = new KeyValueConfiguration(new KeyValueCollection());
config.setTLSEnabled(true);
config.setTLSVersion(TLSConfiguration.TLS_VERSION_1_1);
Endpoint ep = new Endpoint(host,  port, config);
ExternalServiceProtocol protocol = new ExternalServiceProtocol(ep);
```

2) Using the `TLSConfiguration` class:

```
String tlsVersion = TLSConfiguration.TLS_VERSION_1;
TLSConfiguration config = new TLSConfiguration();
config.setVersion(tlsVersion);
SSLContext sslContext = TLSConfigurationHelper.createSslContext(config);
SSLExtendedOptions sslOptions = TLSConfigurationHelper.createSslExtendedOptions(config);
ExternalServiceProtocol protocol = new ExternalServiceProtocol(new Endpoint("TLSClient",
host, port, connectionConfiguration, true, sslContext, sslOptions));
```

3) Using the `SSLContextHelper` and `TLSConfigurationHelper` classes:

```
KeyManager keyManager = new KeyManager() {  };
KeyManager[] keyManagers = {keyManager};
TrustManager trustManager = new TrustManager() { };
TrustManager[] trustManagers = {trustManager};
```

```
SecureRandom secureRandom = new SecureRandom();
SSLContextHelper sslContextHelper = new SSLContextHelper();
SSLContext sslContext = sslContextHelper.createSSLContext(keyManagers, trustManagers,
secureRandom, TLSConfiguration.TLS_VERSION_DEFAULT);
```

4) Using system property settings:

```
System.setProperty("com.genesyslab.platform.commons.connection.tlsDefaultVersion",
TLSConfiguration.TLS_VERSION_1_1);

PsdkCustomization.setOption(PsdkOption.PsdkTlsDefaultVersion,
TLSConfiguration.TLS_VERSION_1_2);
```

.NET

The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications. Since there are various versions of TLS (1.0, 1.1, 1.2, and any future versions) and SSL (2.0 and 3.0), there needs to be a way to negotiate which specific protocol version to use. The TLS protocol provides a built-in mechanism for version negotiation so as not to bother other protocol components with the complexities of version selection.

Tip

The list of available TLS protocol versions, with respect to .NET Framework versions, is provided below:

1. .NET Framework 3.5, 4.0: Ssl2, Ssl3, Tls (TLS 1.0).
2. .NET Framework 4.5.x, 4.6.x, 4.7.x: Ssl2, Ssl3, Tls (TLS 1.0), Tls11 (TLS 1.1), Tls12 (TLS 1.2).

For all .NET Framework versions, setting the TLS protocol version to `SslProtocols.Default` (which is the initial value) means that both SSL 3.0 and TLS 1.0 are acceptable for secure communications.

Platform SDK for .NET includes extended TLS support so that you can specify which version of TLS to use. There are several ways to accomplish this, as described below.

1) Using the `TlsSupport` class.

Server side:

```
var serverSslProtocols = SslProtocols.Tls;
var server = new ServerChannel(new Endpoint(host, port), new
ExternalServiceProtocolFactory());
server.TlsProperties.EnabledSslProtocols = serverSslProtocols;
```

Client side:

```
var clientSslProtocols = SslProtocols.Tls12;
var protocol = new ExternalServiceProtocol(new Endpoint(host, port));
protocol.TlsProperties.EnabledSslProtocols = clientSslProtocols;
```

2) Using the application configuration file:

```
<configuration>
  <appSettings>
    <add key="DefaultClientTLSVersion" value="TLS12 , SSL3"/>
    <add key="DefaultServerTLSVersion" value="tls11"/>
  </appSettings>
</configuration>
```

3) Using the `PsdkCustomization` class:

```
PsdkCustomization.TLSServerVersion.Value = SslProtocols.Tls12;
PsdkCustomization.TLSClientVersion.Value = SslProtocols.Tls11;
```