



This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

Platform SDK Developer's Guide

Using the Application Template Application Block

12/12/2025

Contents

- 1 Using the Application Template Application Block
 - 1.1 Introduction
 - 1.2 Parsing Configuration Objects
 - 1.3 Customizing TLS Configuration
 - 1.4 Creating SSLContext Objects
 - 1.5 Configuring TLS for Client Connections
 - 1.6 Configuring TLS for Servers

Using the Application Template Application Block

Introduction

Instead of [using the Platform SDK Commons Library](#) to configure TLS connections with hard-coded values, you can use the Platform SDK Application Template Application Block to retrieve configuration objects from Configuration Server which contain parameters that are used to configure your TLS settings.

The steps to accomplish this are as follows:

1. Parse a configuration object.
2. Create a `TLSConfiguration` object for the configuration object.
3. Customize your `TLSConfiguration` object:
 - Add callback handlers.
 - For clients, set the expected host names for primary and backup servers.
4. Create `SSLContext` and `SSLEXTENDEDOptions` objects based on your `TLSConfiguration` object.
5. Use your `SSLContext` and `SSLEXTENDEDOptions` objects to create Endpoints and/or `WarmStandbyConfiguration` objects.
6. Use your Endpoints and/or `WarmStandbyConfiguration` objects to create Protocol instances.

The sections below describe these steps in more detail. If you plan on using this method to configure TLS settings, be sure that related application objects in Configuration Manager have been [configured with TLS parameters](#).

Note: If you aren't familiar with TLS configuration settings then please read [Using the Platform SDK Commons Library](#) to gain a better understanding of what is required.

Parsing Configuration Objects

The Platform SDK Application Template has a helper class, `GConfigTlsPropertyReader`, which makes it easy to extract TLS parameters from Configuration Server. When used in conjunction with `TLSConfigurationParser`, `TLSConfigurationHelper`, `ClientConfigurationHelper` and `ServerConfigurationHelper` classes, all of the connection-related options found in Configuration Server are covered. They also provide other useful functionality.

`TLSConfigurationParser` has two constructors:

```
public GConfigTlsPropertyReader(  

```

```
IGApplicationConfiguration appConfig,  
IGApplicationConfiguration.IGPortInfo portConfig);
```

and

```
public GConfigTlsPropertyReader(  
    IGApplicationConfiguration appConfig,  
    IGApplicationConfiguration.IGAppConnConfiguration connConfig);
```

The first one is used for server-side connections while the second is for client-side connections.

For example:

```
// Client side  
// Prepare configuration objects  
String clientAppName = "<my-app-name>";  
CfgAppType targetServerType = CfgAppType.CFGTServer;  
CfgApplication cfgApplication = confService.retrieveObject(  
    CfgApplication.class, new CfgApplicationQuery(clientAppName));  
GCOMApplicationConfiguration appConfiguration =  
    new GCOMApplicationConfiguration(cfgApplication);  
IGApplicationConfiguration.IGAppConnConfiguration connConfig =  
    appConfiguration.getAppServer(targetServerType);  
  
// Parse TLS parameters  
PropertyReader reader = new GConfigTlsPropertyReader(appConfiguration, connConfig);  
TLSConfiguration tlsConfiguration =  
    TLSConfigurationParser.parseTlsConfiguration(reader, true);  
// At this point, tlsConfiguration contains TLS parameters read from  
// configuration objects  
  
// Server side  
// Prepare configuration objects  
String serverAppName = "<my-app-name>";  
String portID = "secure";  
CfgApplication cfgApplication = confService.retrieveObject(  
    CfgApplication.class, new CfgApplicationQuery(serverAppName));  
GCOMApplicationConfiguration appConfiguration =  
    new GCOMApplicationConfiguration(cfgApplication);  
IGApplicationConfiguration.IGPortInfo portConfig =  
    appConfiguration.getPortInfo(portID);  
  
// Parse TLS parameters  
PropertyReader reader = new GConfigTlsPropertyReader(appConfiguration, portConfig);  
TLSConfiguration tlsConfiguration =  
    TLSConfigurationParser.parseTlsConfiguration(reader, false);
```

Customizing TLS Configuration

When Configuration objects are used as a source of TLS parameters, they can also provide values for expected host names.

Examples:

```
TLSConfiguration tlsConfiguration = ...;  
  
// Client side  
// Prepare configuration objects
```

```
String clientAppName = "<my-app-name>";
CfgAppType targetServerType = CfgAppType.CFGTServer;
CfgApplication cfgApplication = confService.retrieveObject(
    CfgApplication.class, new CfgApplicationQuery(clientAppName));
GCOMApplicationConfiguration appConfiguration =
    new GCOMApplicationConfiguration(cfgApplication);
IGApplicationConfiguration.IGAppConnConfiguration connConfig =
    appConfiguration.getAppServer(targetServerType);

// TLS-specific part
IGApplicationConfiguration.IGServerInfo primaryServer =
    connConfig.getTargetServerConfiguration().getServerInfo();
IGApplicationConfiguration.IGServerInfo backupServer =
    primaryServer.getBackup().getServerInfo();

tlsConfiguration.setExpectedHostname(primaryServer.getHost().getName());
// Or:
// tlsConfiguration.setExpectedHostname(backupServer.getHost().getName());
```

Creating SSLContext Objects

SSLContext and SSLEXTENDEDOptions are created either using TLSConfigurationHelper or with TLSConfiguration shortcut methods:

Examples:

```
SSLContext sslContext =
    TLSConfigurationHelper.createSslContext(tlsConfiguration);
SSLEXTENDEDOptions sslOptions =
    TLSConfigurationHelper.createSslExtendedOptions(tlsConfiguration);

// The same as above, using shortcut methods:
sslContext = tlsConfiguration.createSslContext();
sslOptions = tlsConfiguration.createSslExtendedOptions();
```

Configuring TLS for Client Connections

Platform SDK has a helper class, ClientConfigurationHelper, that makes it easier to prepare connections for client applications. This class has the following methods:

```
public static Endpoint createEndpoint(
    IGApplicationConfiguration appConfig,
    IGAppConnConfiguration connConfig,
    IGApplicationConfiguration targetServerConfig);

public static Endpoint createEndpoint(
    IGApplicationConfiguration appConfig,
    IGAppConnConfiguration connConfig,
    IGApplicationConfiguration targetServerConfig,
    boolean tlsEnabled,
    SSLContext sslContext,
    SSLEXTENDEDOptions sslOptions);

public static WarmStandbyConfiguration createWarmStandbyConfig(
    IGApplicationConfiguration appConfig,
```

```
        IAppConnConfiguration connConfig);

public static WarmStandbyConfiguration createWarmStandbyConfig(
    IApplicationConfiguration appConfig,
    IAppConnConfiguration connConfig,
    boolean primaryTLSEnabled,
    SSLContext primarySSLContext,
    SSLExtendedOptions primarySSLOptions,
    boolean backupTLSEnabled,
    SSLContext backupSSLContext,
    SSLExtendedOptions backupSSLOptions);
```

Two of these methods simply accept TLS-specific parameters and pass them through to the Endpoint and WarmStandbyConfiguration instances being created. A code sample using the createEndpoint() method is shown here:

```
String clientAppName = "<my-app-name>";
CfgAppType targetServerType = CfgAppType.CFGTServer;
CfgApplication cfgApplication = confService.retrieveObject(
    CfgApplication.class, new CfgApplicationQuery(clientAppName));

GCOMApplicationConfiguration appConfiguration =
    new GCOMApplicationConfiguration(cfgApplication);

IAppConnConfiguration connConfig =
    appConfiguration.getAppServer(targetServerType);

// TLS preparation section follows
PropertyReader reader = new GConfigTlsPropertyReader(appConfiguration, connConfig);
TLSConfiguration tlsConfiguration =
    TLSConfigurationParser.parseTlsConfiguration(reader, true);

// TLS customization code goes here...
// As an example, host name verification is turned on
IApplicationConfiguration.IGServerInfo targetServer =
    connConfig.getTargetServerConfiguration().getServerInfo();
tlsConfiguration.setExpectedHostname(targetServer.getHost().getName());

// Get TLS configuration objects for connection
SSLContext sslContext = tlsConfiguration.createSslContext();
SSLExtendedOptions sslOptions = tlsConfiguration.createSslExtendedOptions();
boolean tlsEnabled = tlsConfiguration.isTlsEnabled();
// TLS preparation section ends

Endpoint epTSrv = ClientConfigurationHelper.createEndpoint(
    appConfiguration, connConfig,
    connConfig.getTargetServerConfiguration(),
    tlsEnabled, sslContext, sslOptions);

TServerProtocol tsProtocol = new TServerProtocol(epTSrv);
tsProtocol.setClientName(clientName);
tsProtocol.open();
```

Configuring Warm Standby

In cases when the target server has a backup in warm standby mode, configuration requires a little extra effort, as shown in the following code sample.

Note: Configuring TLS for primary and backup servers in Warm Standby mode has some specifics that may not be obvious. Primary and backup servers typically share the same settings. Thus, when a

server is selected as a backup for another server (the primary server), Configuration Manager copies settings from the primary server to the backup server to make them the same. This is also true of TLS settings, and the same TLSConfiguration object can be used to configure both the primary and backup connections. On the other hand, primary and backup servers usually reside on different hosts. This means that if a hostname check is used, each of these servers must have different expectedHostname parameter values. This is not hard to do, as the following code sample demonstrates, but it is not always obvious.

```
String clientAppName = "<my-app-name>";
CfgAppType targetServerType = CfgAppType.CFGStatServer;
CfgApplication cfgApplication = confService.retrieveObject(
    CfgApplication.class, new CfgApplicationQuery(appName));

GCOMApplicationConfiguration appConfiguration =
    new GCOMApplicationConfiguration(cfgApplication);

IGAppConnConfiguration connConfig =
    appConfiguration.getAppServer(targetServerType);

// TLS preparation section follows
PropertyReader reader = new GConfigTlsPropertyReader(appConfiguration, connConfig);
TLSConfiguration tlsConfiguration =
    TLSConfigurationParser.parseTlsConfiguration(reader, true);

IGApplicationConfiguration.IGServerInfo primaryServer =
    connConfig.getTargetServerConfiguration().getServerInfo();
IGApplicationConfiguration.IGServerInfo backupServer =
    primaryServer.getBackup().getServerInfo();

// Configure TLS for Primary
tlsConfiguration.setExpectedHostname(primaryServer.getHost().getName());
SSLContext primarySslContext = tlsConfiguration.createSslContext();
SSLExtendedOptions primarySslOptions = tlsConfiguration.createSslExtendedOptions();
boolean primaryTlsEnabled = tlsConfiguration.isTlsEnabled();

// Configure TLS for Backup
tlsConfiguration.setExpectedHostname(backupServer.getHost().getName());
SSLContext backupSslContext = tlsConfiguration.createSslContext();
SSLExtendedOptions backupSslOptions = tlsConfiguration.createSslExtendedOptions();
boolean backupTlsEnabled = tlsConfiguration.isTlsEnabled();
// TLS preparation section ends

WarmStandbyConfiguration wsConfig =
    ClientConfigurationHelper.createWarmStandbyConfig(
        appConfiguration, connConfig,
        primaryTlsEnabled, primarySslContext, primarySslOptions,
        backupTlsEnabled, backupSslContext, backupSslOptions);

StatServerProtocol statProtocol =
    new StatServerProtocol(wsConfig.getActiveEndpoint());
statProtocol.setClientName(clientName);

WarmStandbyService wsService = new WarmStandbyService(statProtocol);
wsService.applyConfiguration(wsConfig);
wsService.start();
statProtocol.beginOpen();
```

Configuring TLS for Servers

Platform SDK has a helper class, `ServerConfigurationHelper`, that makes it easier to prepare listening sockets for server applications. This class has the following methods:

```
public static Endpoint createListeningEndpoint(
    IApplicationConfiguration application,
    IApplicationConfiguration.IPortInfo portInfo);

public static Endpoint createListeningEndpoint(
    IApplicationConfiguration application,
    IApplicationConfiguration.IPortInfo portInfo,
    boolean tlsEnabled,
    SSLContext sslContext,
    SSLEXTENDEDOptions sslOptions);
```

The overloaded version of the `createListeningEndpoint()` method accepts TLS parameters and passes them through to the `Endpoint` object that is being created. The following code sample shows how this is done:

```
String serverAppName = "<my-app-name>";
String portID = "secure";
CfgApplication cfgApplication = confService.retrieveObject(
    CfgApplication.class, new CfgApplicationQuery(appName));
GCOMApplicationConfiguration appConfig =
    new GCOMApplicationConfiguration(cfgApplication);
IApplicationConfiguration.IPortInfo portConfig =
    appConfig.getPortInfo(portID);

// TLS preparation section follows
PropertyReader reader = new GConfigTlsPropertyReader(appConfiguration, portConfig);
TLSConfiguration tlsConfiguration =
    TLSConfigurationParser.parseTlsConfiguration(reader, false);

// TLS customization code goes here...
// As an example, mutual TLS mode is turned on
tlsConfiguration.setMutual(true);

// Get TLS configuration objects for connection
SSLContext sslContext = tlsConfiguration.createSslContext();
SSLEXTENDEDOptions sslOptions = tlsConfiguration.createSslExtendedOptions();
boolean tlsEnabled = tlsConfiguration.isTlsEnabled();
// TLS preparation section ends

Endpoint endpoint = ServerConfigurationHelper.createListeningEndpoint(
    appConfig, portConfig,
    tlsEnabled, sslContext, sslOptions);
ExternalServiceProtocolListener serverChannel =
    new ExternalServiceProtocolListener(endpoint);
...
```