

# Outbound HTTPS Sample Service

## Introduction

The Outbound HTTPS Sample Service demonstrates a way to create an HTTPS outbound connection using an SSLContext retrieved by the SSLUtilFactory. It also shows the logic needed for a snap-in to be advised of a change to trusted certificate(s) on the platform.

## Concepts demonstrated

1. Outbound HTTPS invocations using Apache HttpClient library.
2. Certificate change listener. When the trusted or identity certificates change, it is important to get a new SSLContext that leverages the new certificate(s).
3. SSLUtilityFactory. This class allows a snap-in to create a SSLContext object that uses the configured Avaya Breeze® platform trusted certificates and identity certificates.
4. Using the Security network interface for outbound requests instead of the Management network interface.

## Detailed description

1. The snap-in creates an HTTPS outbound connection to the Zang cloud server to get the account details and display the account detail.
  - i. The Apache HttpClient library is used for HTTP connections to Zang Cloud.
  - ii. The Avaya Breeze® platform SSLUtilityFactory class is used to create the SSL Context.
2. For the HTTPS outbound connection to work, the Zang cloud server or CA certificate must be added to the Avaya Breeze® platform truststore. *TM*
  - i. Refer to *Administering Avaya Breeze* to add a certificate to the Avaya Breeze® platform trust store.
  - ii. Importing certificates required for HTTPS connections to Zang Cloud:
    - a) Download the following certificates:
      - a. [https://support.cloudflare.com/hc/article\\_attachments/360037898732/origin\\_ca\\_ecc\\_root.pem](https://support.cloudflare.com/hc/article_attachments/360037898732/origin_ca_ecc_root.pem)
      - b. Import the CA root certificate
        - i. Save the root certificate from zang.io to a temporary file with the following command

```
openssl s_client -showcerts -connect api.zang.io:443 </dev/null
2>/dev/null|openssl x509 -outform PEM > zangcert.pem
```

- ii. Add zang.io root certificate to truststore
  - a. Go to SMGR > Services > Inventory > Manage Elements
  - b. Select the servers to add the root certificate
  - c. Click Manage Trusted Certificates from the More Actions dropdown list
  - d. Select Store Type SECURITY\_MODULE\_HTTP
  - e. Select Import as PEM certificate
  - f. Copy and paste the content previously saved in *zangcert.pem* to the space
  - g. Commit the change
  - h. Repeat step (b) to (g) for Store Type WEBSHERE

iii. Update the supported TLS version for outbound request from OutboundHttpsSample service

- a. Go to SMGR > Elements > Avaya Breeze > Cluster Administration
- b. Select cluster then Edit
- c. Go to Services tag and select OutboundHttpsSample service
- d. Select TLS12 from Select TLS Version for Selected Snap-in(s) dropdown list
- e. Commit the change

Go to SMGR > Elements > Avaya Breeze >

b) Navigate to the Cluster Administration page, [Home](#) / [Elements](#) / [Avaya](#)

[Breeze](#)® platform.

c) Select the target cluster, then press Certificate Management, and select

Install Trust Certificate (All Avaya Breeze® platform Instances).

d) In the drop down menu, Select Store Type to install trusted certificate, select WEBSHERE. For each of the downloaded certificates: Choose File; select the downloaded certificate; Retrieve Certificate; and Commit. iii. The sample does not show the process by which you can automatically copy the certs for the server that you are trying to connect to during snap-in install.

This is because the Outbound HTTP Sample snap-in is not an Avaya signed application and the process applies only to Avaya signed applications. To automatically copy the certs during installation, you must copy the certs to a folder you create under the

resources folder. The folder you create must be named certs. Then if you publish your snap-in with Avaya Snapp-store (this publishing signs the snap-in), the steps in 2 ii, are not needed. Look at the service archetype in the Avaya Breeze® platform SDK for more details on using this mechanism. Files that will contain comments are dist.xml and pom.xml of the snap-in svar.

3. The snap-in uses the @ThePlatformListener to listen for the events that indicate that the certificate store has been updated. This event triggers the snap-in to update the SSLContext used in the HTTPS outbound connection.

### Mandatory Service Attributes to be configured:

Account ID			String used for account ID display associated with the account created at www.zang.io.
Auth Token			The auth token associated with the account created at www.zang.io.
Phone Number			The Mobile Phone number for which carrier lookup will be performed.

### Installation and configuration

For information on installing the service and configuring the service attributes see *Administering Avaya Breeze® platform*.

## Testing the service

To configure the service attributes you need to have a Zang account, please refer to <http://www.zang.io/> to create the account.

This service contains the below two REST APIs .Both the API's can be invoked by simply putting the URL into the browser.

- 1) View Account details API : To see all the information associated with your account.  
<https://SecurityIPAddress/services/OutboundHttpsSample/ws/myResource/accountDetails>
- 2) The Carrier Lookup API : Allows you to retrieve additional information about a phone.  
<https://SecurityIPAddress/services/OutboundHttpsSample/ws/myResource/carrierLookup>

## Code snippets:

### 1) SSLUtilityFactory to get the SSLContext

The `SSLUtilityFactory.createSSLContext()` in the Avaya Breeze® platform SDK creates the `SSLContext` object. This code snippet is present in `HttpClientSingleton` enum `getSSLContext()` method.

```
SSLUtilityFactory.createSSLContext();
```

### 2) Setting the SSLContext in the HttpClient

The `getHttpClientBasedOnSSLContext()` method creates the `HttpClient` object using the `SSLContext` object created above. This code snippet is present in `HttpClientSingleton` enum.

```
final SSLConnectionSocketFactory
sslConnectionSocketFactory = new
SSLConnectionSocketFactory(
    SSLUtilityFactory.createSSLContext(),
    NoopHostnameVerifier.INSTANCE);
final Registry<ConnectionSocketFactory> registry
=
RegistryBuilder.<ConnectionSocketFactory>
create()
    .register("http", PlainConnectionSocketFactory.getSocketFactory())
    .register("https", sslConnectionSocketFactory)
    .build();
final HttpClientConnectionManager cm = new
BasicHttpClientConnectionManager(registry);
client = HttpClients.custom().setConnectionManager(cm)
    .build();
```

### 3) Using platform listener to get the cert change update

The certificateStoreUpdated () method gets invoked as soon as a certificates is added to the node. Here we are invoking the reset of HttpClientSingleton to create a new HttpClient using the new certificate changes. This code snippet is present in CertificateChangePlatformListener class.

```
@ThePlatformListener
public class CertificateChangePlatformListener extends PlatformListenerAbstract
{
    private Logger logger =
    Logger.getLogger(getClass()); @Override public final void certificateStoreUpdated()
    {
        HttpClientSingleton.INSTANCE.reset();
    }
}
```

### 4) Using the security module interface for outbound connectivity

```
final ZephyrDM dm = (ZephyrDM) DMFactory.getInstance().getDataMgr(
ZephyrDM.class); myFqdnOrIpAddress = dm.getMySIPEntity().getFqdnoripaddr();
requestConfig =
getRequestConfigBuilder().setLocalAddress(InetAddress.getByName(myFqdnOrIpAdress
s)).build();
client = HttpClients.custom().setDefaultRequestConfig(requestConfig).build();
```