# Documento: Uso de la versión 4.0 de la librería PGE

|  |  |  |
| --- | --- | --- |
| Autor | Fecha | Descripción |
| Andrés Pastorini | 17-6-2021 |  |

## Introducción

El presente documento ilustra, por medio de una aplicación de ejemplo cuyos fuentes se encuentran en la carpeta \_fuentes en https://github.com/AGESIC-UY/cliente-java-plataforma-interoperabilidad/tree/v4.0/doc/example, el uso de la librería facilitada por AGESIC para facilitar el consumo de servicios de PDI, cuya última versión se puede obtener [aquí](https://github.com/AGESIC-UY/cliente-java-plataforma-interoperabilidad) .

## Integración librería PGE

Verá que la aplicación contiene la lógica para invocar ciertos servicios, pero al ejecutar los servicios estos no traerán información, ya que por cuestiones de seguridad se omitió la publicación de certificados , trustores y keystores que irían en la carpeta \_fuentes/pge.

En el ejemplo, sobre la librería base se desarrolló un componente que dado un proxy de servicio web, generado con AXIS 1.4 (las distintas librerias utilizadas como sus versiones, se puede obtener del archivo pom.xml ), se encarga de utilizar la librería PGE, obtener un security assertion e insertar el mismo en la instancia del proxy del servicio. De esta forma es casi transparente para el desarrollador, que se está utilizando el servicio a través de PDI.

A modo de ejemplo, a continuación, se muestra cómo ejemplo la invocación a un servicio publicado en PDI.

RUTPersonaGetEntidadLocator locator = **new** RUTPersonaGetEntidadLocator();

RUTPersonaGetEntidadExecute params = **new** RUTPersonaGetEntidadExecute();

RUTPersonaGetEntidadSoapBindingStub stub = **new** RUTPersonaGetEntidadSoapBindingStub(**new** URL(Config.*GetValuePGE*("dgi", Config.*WS\_URL*)),locator);

pgeFacilitaror.addWSSecurity(stub, "dgi");

params.setRuc(rut);

PersonaGetEntidadResponse responsePGE = **null**;

RUTPersonaGetEntidadExecuteResponse result = stub.execute(params);

Como puede observarse, lo único “extra” que se requiere hacer es llamar a un método de pgeFacilitator, pasando la instancia del proxy e indicando el nombre del servicio para el archivo de propiedades (de las cuales obtienen diversos datos necesarios para obtener un security assertion.

A continuación, se muestran las funciones auxiliares que se definieron y que son las que finalmente terminan interactuando contra la librería PGE.

public void addWSSecurity(org.apache.axis.client.Stub stub, String servicio) throws SOAPException, TransformerException {

String wsaTo = Config.GetValuePGE(servicio, Config.WS\_WSA\_TO);

String wsaAction = Config.GetValuePGE(servicio, Config.WS\_WSA\_ACTION);

String stsService = Config.GetValuePGE(servicio, Config.WS\_SERVICE);

Element assertion = getWSSecurity(servicio,wsaTo,wsaAction, stsService);

for (int i=0; i<assertion.getChildNodes().getLength(); i++)

{

Element e = (Element)assertion.getChildNodes().item(i);

SOAPHeaderElement header0 = new SOAPHeaderElement(e);

stub.setHeader(header0);

}

}

private Element getWSSecurity(String servicio, String wsaTo, String wsaAction, String stsService) {

boolean exito = false;

if (wsaTo == null || wsaAction == null || stsService == null) {

return null;

}

String saml1Property = Config.GetValuePGE(Config.SAML\_PROPERTY);

String stsURL = Config.GetValuePGE(Config.STS\_URL);

String stsIssuer = Config.GetValuePGE(Config.STS\_ISSUER);

String stsPolicyName = Config.GetValuePGE(Config.STS\_POLICY\_NAME);

String stsRole = Config.GetValuePGE(servicio + "."+ Config.WS\_STS\_ROLE);

if(stsRole == null || stsRole == "")

stsRole = Config.GetValuePGE(Config.STS\_ROLE);

String stsUserName = Config.GetValuePGE(servicio + "."+ Config.WS\_STS\_USERNAME);

if(stsUserName == null || stsUserName == "")

stsUserName = Config.GetValuePGE(Config.STS\_USERNAME);

//String stsService = Config.GetValuePGE(Config.STS\_SERVICE);

String stsSecurityActor = Config.GetValuePGE(Config.STS\_SECURITY\_ACTOR);

String stsKeyStoreFilePath = Config.GetValuePGE(Config.STS\_KEYSTORE\_FILEPATH);

String stsKeyStorePass = Config.GetValuePGE(Config.STS\_KEYSTORE\_PASS);

String stsKeyStoreCertAlias = Config.GetValuePGE(Config.STS\_KEYSTORE\_ALIAS);

String sslClientKeyStoreFilePath = Config.GetValuePGE(Config.SSL\_CLIENT\_KEYSTORE\_FILEPATH);

String sslClientKeyStorePass = Config.GetValuePGE(Config.SSL\_CLIENT\_KEYSTORE\_PASS);

String sslClientKeyStoreCertAlias = Config.GetValuePGE(Config.SSL\_CLIENT\_KEYSTORE\_ALIAS);

String sslTrustStorePass = Config.GetValuePGE(Config.SSL\_TRUSTSTORE\_PASS);

String sslTrustStoreFilePath = Config.GetValuePGE(Config.SSL\_TRUSTSTORE\_FILEPATH);

logger.debug("lectura de configuración finalizada");

Element security = null;

try {

security = getWSSecuritySOAP(wsaTo, wsaAction, saml1Property, stsURL, stsIssuer, stsPolicyName,

stsRole, stsUserName, stsService, stsSecurityActor, stsKeyStoreFilePath, stsKeyStorePass,

stsKeyStoreCertAlias, sslClientKeyStoreFilePath, sslClientKeyStorePass, sslClientKeyStoreCertAlias,

sslTrustStoreFilePath, sslTrustStorePass);

} catch (XMLStreamException e) {

} catch (Exception e) {

}

return security;

}

public Element getWSSecuritySOAP(String wsaTo, String wsaAction, String saml1Property, String stsURL,

String stsIssuer, String stsPolicyName, String stsRole, String stsUserName, String stsService, String stsSecurityActor,

String stsKeyStoreFilePath, String stsKeyStorePass,

String stsKeyStoreCertAlias, String sslClientKeyStoreFilePath,

String sslClientKeyStorePass, String sslClientKeyStoreCertAlias,

String sslTrustStoreFilePath, String sslTrustStorePass) throws XMLStreamException, KeyStoreException, FileNotFoundException, RequestSecurityTokenException, KeyManagementException, UnrecoverableKeyException, NoSuchAlgorithmException{

Configuration config = new Configuration(stsKeyStoreCertAlias, stsKeyStorePass, stsKeyStorePass,

sslTrustStorePass, stsKeyStoreFilePath, sslClientKeyStoreFilePath, sslTrustStoreFilePath);

RoleOperation role = new RoleOperation();

role.setRole(stsRole);

role.setWsaAction(wsaAction);

role.setSoapVersion("1.1");

Connector connector = new Connector(role, wsaTo,stsUserName, stsIssuer,false,false,"");

CloseableHttpClient httpClient = new DefaultHttpClient();

KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());

FileInputStream instream = new FileInputStream(new File(sslClientKeyStoreFilePath));

try {

keyStore.load(instream, sslClientKeyStorePass.toCharArray());

}catch(NoSuchAlgorithmException e){

logger.error(e.getMessage());

} catch(CertificateException e){

} catch (IOException e) {

} finally{

try{

instream.close();

}catch(Exception ignore){

logger.error(ignore.getMessage());

}

}

KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());

instream = new FileInputStream(new File(sslTrustStoreFilePath));

try{

trustStore.load(instream, sslTrustStorePass.toCharArray());

}catch(NoSuchAlgorithmException e){

logger.error(e.getMessage());

}catch(CertificateException e){

logger.error(e.getMessage());

} catch (IOException e) {

logger.error(e.getMessage());

}finally{

try{

instream.close();

}catch(Exception ignore){

logger.error(ignore.getMessage());

}

}

SSLSocketFactory socketFactory = new SSLSocketFactory(keyStore, sslClientKeyStorePass, trustStore);

Scheme sch = new Scheme("https", 443, socketFactory);

httpClient.getConnectionManager().getSchemeRegistry().register(sch);

PGEClient client = new PGEClientCache(new PGEClientBasic());

STSResponse stsResponse = client.requestSecurityToken(config, connector, stsPolicyName,

stsURL, httpClient);

Document d = convertStringToXMLDocument(WS\_SECURED\_HEADER\_PART);

Element header = (Element)d.getDocumentElement();

header.getFirstChild().appendChild(d.importNode(stsResponse.getAssertion().getDOM(),true));

header.appendChild(d.importNode(convertStringToXMLDocument(getWSAddressingToPart(wsaTo))

.getDocumentElement(),true));

header.appendChild(d.importNode(convertStringToXMLDocument(getWSAddressingActionPart(wsaAction))

.getDocumentElement(),true));

logger.debug("soap assertion obtenido");

return (Element) d.getDocumentElement();

}