MAGYAR
TURISZTIKAI ÜGYNÖKSÉG

PMS Interface C# client guide

Version 8.00

Implementing a National Tourism Database (NTAK)

Project ID: 2017/S 249-525381

25.11.2020.

1. Document Control

1.1. Document data

Document Title	PMS Interface C# client guide
Project Name	Implementing a National Tourism Database
Author(s)	Gábor Szmetankó
Responsible Person	Tamás Fekete
Electronic File Name	NTAK_PMS_Interface_CSharp_Client_Guide_v2.00_ENG.docx
Document Version	V2.00
Number of Pages	16
Status	Request for comment

1.2. Version history

Version	Status	Date	Responsible Person	Reason for change
v1.00		08/07/2019	HTA	Initial version
v1.01		12/12/2019	HTA	Updated sample to deal with more strict validation rules
V2.00		25/11/2020	HTA	Update to v8

Contents

2. Introduction

As part of the implementation of the National Tourism Database the Hungarian Tourism Agency released a document called PMS Interface Description, which described the prerequisites for integrating the NTAK system with the PMS software, the necessary configuration for integration on the PMS side, as well as its communication channels and the rules for data transmission.

Included in that document were code samples that were meant to provide a starting point for implementing the integration with the NTAK system. The primary purpose of those code examples was to guide the PMS developers through the first steps of the integration. Unfortunately, the C# example code was used as the basis for several PMS integrations, which lead to incorrect functionality due to the incomplete nature of the code snippets.

This document provides a more detailed introduction and guidance for the implementation of C# based PMS product integrations. The most frequently asked questions and their answers are also included in this document. Attached to this document is a complete Visual Studio Solution, which helps the PMS developer to make the first steps.

It is important to note that the examples in this document are still not suitable for implementing production grade applications. Implementing proper error handling and clean structuring of the code is still the responsibility of the PMS developer.

3. Microsoft .NET-based implementation example

As also mentioned in the introductory sections, the technical prerequisites for successful accession to the NTAK System include, for one, the use of suitable authentication and signature certificates during data transmission, and on the other hand, for messaging to include the appropriate SOAP data structure. The following steps will be required to meet that prerequisite in the case of Microsoft .NET-based implementation:

1. Generate signature and authentication key-pair (PEM)
2. Create certificate request based on the key-pair (CSR)
3. Upload certificate request to the NTAK System
4. Download certificates issued by the NTAK System (CER)
5. Place key-pairs and certificates in a password-protected container (P12)
6. Create client classes for the NTAK endpoints based on the WSDL files
7. Develop HTTPS connectivity to the NTAK endpoints using the authentication certificate
8. Electronic signature of SOAP envelopes posted from the NTAK endpoints using the signature certificate
9. One-way and non-decompilable transformation of guest ID details using the BCrypt algorithm

Although providing the above steps is possible along the use of multiple components, the following ones were used in this section:

1. BouncyCastle.NetCore component.
   For generating the certificate request, and placing the certificate and key-pair in a password-protected container
2. BCrypt.Net-Next component.
   One-way and non-decompilable transformation of guest ID details
3. Microsoft.Web.Services3 component.
   Electronic signature of posted SOAP envelopes

The code sections described in the following subsections provide help with the execution of the necessary steps.

3.1. Generate signature and authentication key-pair

The 4096-bit length of the key, and the RSA algorithm used needs to be specified to generate the key-pair consisting of private and public keys.

```
// Generate 4096-bit long RSA key-pair
var random = new SecureRandom();
var parameters = new KeyGenerationParameters(random, 4096);
var generator = new RsaKeyPairGenerator();
generator.Init(parameters);
var keypair = generator.GenerateKeyPair();
```

Saving the key-pair thus generated is recommended for subsequent use

```
FileStream file = new FileStream("reg00008.pem", FileMode.Create, FileAccess.Write,
FileShare.Read);
StreamWriter streamWriter = new StreamWriter(file);
PemWriter pemWriter = new PemWriter(streamWriter);
pemWriter.WriteObject(keypair);
streamWriter.Close();
```

It is important to note that the above generation of the key pair and the CSR needs to be done twice. Once for the authentication and once for the signing certificate.

3.2. Create certificate request based on the key-pair

The Subject field on the certificate must be completed correctly to generate the certificate

```
string szallasRegisztraciosSzam = "reg00008";
string szallasNev = "Vadvirág Panzió";
string szallasIranyitoszam = "3325";
string szallasTelepules = "Noszvaj";
string szallashelySzolgaltatoAdoszam = "12345678";

// Generate certificate request
//
// The value of the subjectName field has to be populated with the accommodation’s details
// Common Name (CN) – The name of the Subject: The accommodation’s registration number
// Organization (O) – The name of the Organization: Accommodation name
// Organization Identifier (OrgId) – OID: 2.5.4.97 – Organisation ID: The accommodation
services provider’s tax number.
// Country (C) – Country identifier: Hungary, i.e. the ‘HU’ value is to be entered here in
every case
// Locality Name (L) – Settlement name: Accommodation settlement
// Postal Code – OID: 2.5.4.17 – Postcode: Accommodation postcode

var subjectName = $"CN={szallasRegisztraciosSzam}, O={szallasNev},
2.5.4.97={szallashelySzolgaltatoAdoszam}, 2.5.4.17={szallasIranyitoszam},
L={szallasTelepules}, C=HU";
var subject = new X509Name(subjectName);
var factory = new Asn1SignatureFactory(PkcsObjectIdentifiers.Sha512WithRsaEncryption.Id,
keypair.Private, random);
```

```
var request = new Pkcs10CertificationRequest(factory, subject, keypair.Public, null,
keypair.Private);
```

The certificate request thus created must be saved to a file and sent to the NTAK System.

```
FileStream file = new FileStream("reg00008.csr", FileMode.Create, FileAccess.Write,
FileShare.Read);
StreamWriter streamWriter = new StreamWriter(file);
PemWriter pemWriter = new PemWriter(streamWriter);
pemWriter.WriteObject(request);
streamWriter.Close();
```

3.3. Place key-pairs and certificates in a password-protected container

Placing the certificate issued by the NTAK System and the previously generated key-pair in a password-protected container is recommended, as are appropriate safeguards for both the container file and the password linked to it.

```
// Read certificate
X509CertificateParser parser = new X509CertificateParser();
FileStream file = new FileStream("reg00008.cer", FileMode.Open, FileAccess.Read,
FileShare.Read);
X509Certificate certificate = parser.ReadCertificate(file);
file.Close();

// Read key-pair
file = new FileStream("reg00008.pem", FileMode.Open, FileAccess.Read, FileShare.Read);
StreamReader streamReader = new StreamReader(file);
PemReader pemReader = new PemReader(streamReader);
AsymmetricCipherKeyPair keypair = (AsymmetricCipherKeyPair) pemReader.ReadObject();
streamReader.Close();

// Create key container
var random = new SecureRandom();
string keyPassword = "1111";
Pkcs12Store store = new Pkcs12StoreBuilder().Build();

// Upload key container
X509CertificateEntry certificateEntry = new X509CertificateEntry(certificate);
store.SetCertificateEntry("reg00008", certificateEntry);
```

```
AsymmetricKeyEntry keyEntry = new AsymmetricKeyEntry(keypair.Private);
store.SetKeyEntry("reg00008", keyEntry, new X509CertificateEntry[] { certificateEntry });

// Save key container
file = new FileStream("reg00008.p12", FileMode.Create, FileAccess.Write, FileShare.Read);
store.Save(file, keyPassword.ToCharArray(), random);
file.Close();
```

3.4. Create client classes for SOAP endpoints

The classes and data sets that can be used to complete the SOAP envelope to be sent can be created on the basis of the SOAP endpoints' WSDL definition files.

```
wsdl.exe" /out:NTAK.cs /nologo /order /protocol:SOAP
          /namespace:NTAK /sharetypes /enableDataBinding
          esemenyvezerelt-adatkuldes.wsdl
          napi-zaras-teszt.wsdl
          napi-zaras.wsdl
          napi-zaras-utemezes.wsdl
```

It is possible that newer .NET framework versions do not create client classes on the basis of the WebServicesClientProtocol class corresponding to the Microsoft.Web.Services3 component, but using the SoapHttpClientProtocol base class instead. In this case the NTAK.cs file that was created needs to have all the

```
System.Web.Services.Protocols.SoapHttpClientProtocol
```

base classes replaced with the corresponding

```
Microsoft.Web.Services3.WebServicesClientProtocol
```

base class

3.5. Create a SoapFilter

Due to an issue in the WSE3 library, correctly issued certificates sometimes cannot be used for signing documents. This issue only exists in applications that use the WSE3 library. In order to sign the SOAP envelope correctly, we need to extend the default WSE3 solution with a custom SoapFilter, which will attach the digital signature to the outgoing messages. In order to achieve this, we first need to create a SecurityPolicyAssertion subclass.

```
class SigningPolicyAssertion : SecurityPolicyAssertion
{
```

```
   private Security security;

   public SigningPolicyAssertion(Security security)
   {
      this.security = security;
   }

   public override SoapFilter CreateClientInputFilter(FilterCreationContext context)
   {
      return null;
   }

   public override SoapFilter CreateClientOutputFilter(FilterCreationContext context)
   {
      return new ClientOutputFilter(this, security);
   }

   public override SoapFilter CreateServiceInputFilter(FilterCreationContext context)
   {
      return null;
   }

   public override SoapFilter CreateServiceOutputFilter(FilterCreationContext context)
   {
      return null;
   }
}
```

Then, we need to activate it in the NTAK.cs file's constructor:

```
public napiZarasUtemezesPortv8Soap11()
{
    this.Url = "http://192.168.9.146:8080/ntak/v8";
    // Adding the signing policy to the client.
    Policy policy = new Policy();
    policy.Assertions.Add(new SigningPolicyAssertion(RequestSoapContext.Security));
    this.SetPolicy(policy);
}
```

3.6. Applying the authentication certificate

Client-side certificates should be used when the HTTPS communication channel for SOAP endpoints is configured.

```
// Read authentication key
String keyPath = "reg00008-authentication.p12";
String keyPassword = "1111";
X509Certificate2 certificate = new X509Certificate2(keyPath, keyPassword);
```

```
// Client class structure
var client = new NTAK.napiZarasUtemezesPortv8Soap11();
client.Url = "https:// ... ";
client.ClientCertificates.Add(certificate);
```

3.7. Applying the signing certificate

The signing certificate must be applied to the Timestamp and Body elements in order to sign the SOAP messages electronically. This is done by the ClientOuputFilter referenced in SecurityPolicyAssertion.

```
internal class ClientOutputFilter : SoapFilter
{
    private readonly Security security;

    public ClientOutputFilter(SigningPolicyAssertion parentAssertion, Security security)
    {
        this.security = security;
    }
    public override SoapFilterResult ProcessMessage(SoapEnvelope envelope)
    {
        var actionNode = envelope.Header.GetElementsByTagName("Action",
"http://schemas.xmlsoap.org/ws/2004/08/addressing");
        var messageIdNode = envelope.Header.GetElementsByTagName("MessageID",
"http://schemas.xmlsoap.org/ws/2004/08/addressing");
        var replyToNode = envelope.Header.GetElementsByTagName("ReplyTo",
"http://schemas.xmlsoap.org/ws/2004/08/addressing");
        var toNode = envelope.Header.GetElementsByTagName("To",
"http://schemas.xmlsoap.org/ws/2004/08/addressing");
        envelope.Header.RemoveChild(actionNode.Item(0));
        envelope.Header.RemoveChild(messageIdNode.Item(0));
        envelope.Header.RemoveChild(replyToNode.Item(0));
        envelope.Header.RemoveChild(toNode.Item(0));

        // Attach an ID attribute to the Body element. This is required for the References.
        XmlAttribute idAttrib = envelope.CreateAttribute("wsu", "Id", "http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd");
        idAttrib.Value = "Body";
        envelope.Body.Attributes.Append(idAttrib);

        // Read the signing certificate
        var keyPath = "alairo.p12";
        var keyPassword = "1111";
        X509Certificate2 certificate = new X509Certificate2(keyPath, keyPassword);

        // WSSE Security element
        var token = new X509SecurityToken(certificate);
        security.Tokens.Add(token);
        security.Timestamp.TtlInSeconds = 43200;
```

```
        security.SerializeXml(envelope);

        // Digital signature
        var signedXml = new SignedXmlWithId(envelope);
        signedXml.SigningKey = certificate.GetRSAPrivateKey();
        signedXml.SignedInfo.SignatureMethod = SignedXml.XmlDsigRSASHA384Url;
        signedXml.SignedInfo.CanonicalizationMethod =
SignedXml.XmlDsigExcC14NTransformUrl;

        AddReference(signedXml, security.Timestamp.Id);
        AddReference(signedXml, "Body");

        KeyInfo keyInfo = new KeyInfo();
        SecurityTokenReference tokenRef = new SecurityTokenReference(token);
        keyInfo.AddClause(tokenRef);
        signedXml.KeyInfo = keyInfo;

        signedXml.ComputeSignature();
        XmlElement signedElement = signedXml.GetXml();

        // Insert the signature into the WSSE Security element of the SOAP Header
        XPathNavigator signatureNavigator = signedElement.CreateNavigator();
        XPathNavigator headerNavigator = envelope.Header.CreateNavigator();
        XmlNamespaceManager namespaceManager = new
XmlNamespaceManager(envelope.NameTable);
        namespaceManager.AddNamespace("wsse", "http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd");
        XPathNavigator secNode = headerNavigator.SelectSingleNode("wsse:Security",
namespaceManager);
        secNode.AppendChild(signatureNavigator);

        return SoapFilterResult.Continue;
    }

    private void AddReference(SignedXml signedXml, string uri)
    {
        Reference reference = new Reference();
        reference.Uri = "#" + uri;
        reference.DigestMethod = SignedXml.XmlDsigSHA384Url;
        XmlDsigExcC14NTransform env = new XmlDsigExcC14NTransform();
        reference.AddTransform(env);

        signedXml.AddReference(reference);
    }
}
```

The above referenced SignedXmlWithId class is the following:

```
// Extension of the SignedXml class in order to accept wsu:Id atrributes as identifiers.
```

```
internal class SignedXmlWithId : SignedXml
{
    public SignedXmlWithId(SoapEnvelope envelope) : base(envelope)
    {
    }

    public override XmlElement GetIdElement(XmlDocument document, string idValue)
    {
        XmlElement idElem = base.GetIdElement(document, idValue);

        if (idElem == null)
        {
            XmlNamespaceManager nsManager = new
XmlNamespaceManager(document.NameTable);
            nsManager.AddNamespace("wsu", "http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd");

            idElem = document.SelectSingleNode("//*[@wsu:Id=\"" + idValue + "\"]",
nsManager) as XmlElement;
        }

        return idElem;
    }
}
```

4. Frequently asked questions

4.1. The signing certificate does not contain the Digital Signature Key Usage extension, therefore the NTAK provided certificates are invalid and cannot be used for digital signatures.

In order to understand the purpose and meaning of the KeyUsage attribute in the certificate, we first need to look at the X.509 standard, especially the relevant section:

```
id-ce-keyUsage OBJECT IDENTIFIER ::=  { id-ce 15 }

KeyUsage ::= BIT STRING {
     digitalSignature        (0),
     nonRepudiation          (1), -- recent editions of X.509 have
                          -- renamed this bit to contentCommitment
     keyEncipherment         (2),
     dataEncipherment        (3),
     keyAgreement            (4),
     keyCertSign             (5),
     cRLSign                 (6),
     encipherOnly            (7),
     decipherOnly            (8) }

Bits in the KeyUsage type are used as follows:

   The digitalSignature bit is asserted when the subject public key
   is used for verifying digital signatures, other than signatures on
   certificates (bit 5) and CRLs (bit 6), such as those used in an
   entity authentication service, a data origin authentication
   service, and/or an integrity service.

   The nonRepudiation bit is asserted when the subject public key is
   used to verify digital signatures, other than signatures on
   certificates (bit 5) and CRLs (bit 6), used to provide a non-
   repudiation service that protects against the signing entity
   falsely denying some action.  In the case of later conflict, a
   reliable third party may determine the authenticity of the signed
   data.  (Note that recent editions of X.509 have renamed the
   nonRepudiation bit to contentCommitment.)
```

As can be seen above, the nonRepudiation KeyUsage value is used for digital signing. This is the mandatory value in official signing certificates.

By looking at the definition of the digitalSignature KeyUsage value, it becomes clear that it is mainly used in authentication scenarios. This is the reason why the NTAK provided authentication certificates contain the digitalSignature KeyUsage value and the signing certificates do not.

For further guidance, it is recommended to look at section 7.4.6 of the TLS1.2 standard (https://tools.ietf.org/html/rfc5246#page-56) as well, which describes the requirement for a client certificate.

Client Cert. Type	Certificate Key Type
rsa_sign	RSA public key; the certificate MUST allow the key to be used for signing with the signature scheme and hash algorithm that will be employed in the certificate verify message.

The relevant section from the ETSI standard also confirms the above statements (https://www.etsi.org/deliver/etsi_en/319400_319499/31941202/02.01.01_60/en_31941202v020101p.pdf):

Table 1: Key usage settings

Type	Non-Repudiation (Bit 1)	Digital Signature (Bit 0)	Key Encipherment or Key Agreement (Bit 2 or 4)
A	X		
B	X	X	
C		X	
D		X	X
E			X
F	X	X	X

Certificates used to validate commitment to signed content shall be limited to type A, B or F. Of these alternatives, type A should be used (see the security note 2 below).

EXAMPLE: Digital signatures on agreements and/or transactions.

NOTE 1: The X.509 standard [i.3] has renamed the nonRepudiation bit to "contentCommitment". IETF RFC 5280 [1] has kept the original name nonRepudiation for backwards compatibility reasons. These bits are equivalent in function and meaning regardless of their different names.

Based on the above, it can be clearly seen that the certificates provided by the NTAK system are in conformance with the standards.

Signing issues are only caused by the incorrectly implemented WSE3 library, which requires signing certificates to include the digitalSignature KeyUsage extension, when it would not be necessary.

4.2. The underlying connection was closed: Could not establish trust relationship for the SSL/TLS secure channel.

The above error message can usually be seen in the test environment. The reason for this message is that in the test environment the NTAK system uses a self-signed certificate, which are not trusted by default. The issue can be resolved by adding the self-signed certificate of the NTAK server (the complete certification chain) to the Trust Store of the machine running the PMS application. This will make the NTAK server a trusted server.

In case of .NET applications, we can do this by opening the Windows Certificate Manager application. In Start/Run, enter certlm.msc and press Enter. In the Certificate Manager click on the Trusted Root Certification Authorities group and import the NTAK certificates. The certificate chain can be easily obtained by loading the https://pms.sgmdev.hu/ntak URL in a browser and exporting the certificates from there.