OIO Service Oriented Infrastructure

OIOSI RASP Library for .NET Version 2.1.0

Tutorials

Contents

1 Introduction

The purpose of this document is to guide developers in creating an application using the RASP library. The main components will be explained here, and example code showing how to set them up will be given.

This document represents a high-level view of the main communication components. More documentation is found

- As comments in the code
- As stand-alone test applications
- As NUnit tests
- By seeing code used as part of other components

Please visit http://www.digst.dk/It-loesninger/NemHandel for more information on the OIOSI project.

2 Overview of Lessons

The basics of the RASP Library will be explained through different lessons, each describing one specific aspect of the API.

The lessons are

- Configuring RASP and Windows Communication Foundation
- Sending documents
- Extended Requests (including UDDI, LDAP and OCSP/CRL lookups)
- Receiving documents
- The interceptors
- Hosting a HTTP service in IIS

3 Prerequisites

3.1 Resources

All common resources can be found in the resource project src/ dk.gov.oiosi.resource, and is added (by link) to other project that need the resources.

3.2 Sample Documents

There are several sample documents used in the tests of the rasp library. They are located in the src\dk.gov.oiosi.raspProfile\Resources\Documents directory.

4 Sending Configuration - App.Config

Each test project has an App.Config application configuration file, and so should every project that uses the RASP Library.

Most of the Windows Communication Foundation settings are done in App.Config, and this section will briefly cover each important section in the configuration.

It is recommended that the main sections described here are copied from one of the test projects, since the RASP Library is dependent on default settings (first and foremost the OiosiHttpEndpoint, OiosiHttpEndpointBehavior, OiosiHttpEndpointBinding and NHR_lookup).

4.1 Client endpoint

App.Config sample 1

```
<system.serviceModel>
   ...
   <client>
     <!-- The OIOSI RASP HTTP endpoint - Sending -->
     <!-- The endpoint name must be "OiosiHttpEndpoint". Hard-code implemented in RASP -->
     <endpoint name="OiosiHttpEndpoint"
               address=""
               binding="customBinding"
               bindingConfiguration="OiosiHttpEndpointBinding"
               contract="dk.gov.oiosi.communication.client.IClientProxyContract"
               behaviorConfiguration="OiosiHttpEndpointBehavior" />
  </client>
  ...
<system.serviceModel>
```

This sample shows a client endpoint, “OiosiHttpEndpoint”, using the attributes

- **name** – Must be OiosiHttpEndpoint, as the name is hardcoded in RASP.
- **address** – Is retrieved from NemHandelsRegisteret, so can be left empty
- **binding** – Must be ‘customBinding’, as it described that we used a custom binding.
- **bindingConfiguration** – The name of the binding to use, described in the customConfiguration section 4.3.
- **contact** – The contact describing the interface.
- **behaviorConfiguration** – The name of the behavior to use, described in section 4.2.

4.2 Client behavior

The behavior configuration for sending.

App.Config sample 2

```
<system.serviceModel>
   ...
  <behaviors>
    <!-- The OIOSI RASP HTTP behavior - Sending -->
    <endpointBehaviors>
      <behavior name="OiosiHttpEndpointBehavior">
        <signCustomHeaders>
          <headers>
            <add name="SenderPartyIdentifier"
namespace="http://rep.oio.dk/oiosi.ehandel.gov.dk/xml/schemas/2007/09/01/" />
            <add name="SenderPartyIdentifierType"
namespace="http://rep.oio.dk/oiosi.ehandel.gov.dk/xml/schemas/2007/09/01/" />
            <add name="ReceiverPartyIdentifier"
namespace="http://rep.oio.dk/oiosi.ehandel.gov.dk/xml/schemas/2007/09/01/" />
            <add name="ReceiverPartyIdentifierType"
namespace="http://rep.oio.dk/oiosi.ehandel.gov.dk/xml/schemas/2007/09/01/" />
            <add name="MessageIdentifier"
namespace="http://rep.oio.dk/oiosi.ehandel.gov.dk/xml/schemas/2007/09/01/" />
          </headers>
        </signCustomHeaders>
      </behavior>
    </endpointBehaviors>
  </behaviors>
  ...
<system.serviceModel>
```

The behavior name (‘`OiosiHttpEndpointBehavior`’) must match the defined name in section 4.1.

4.3 Client binding

The binding configuration for sending.

App.Config sample 3

```
<system.serviceModel>
  ...
  <bindings>
    <basicHttpBinding>
      <!-- Basic http binding is used to perform lookup calls in
NemHandelsRegisteret. -->
      <!-- The endpoint is hard-code implemented in RASP, and the binding must
exist here with the name NHR_lookup -->
      <binding name="NHR_lookup" textEncoding="utf-8"
maxReceivedMessageSize="10000000" />
    </basicHttpBinding>

    <customBinding>
      <!-- The OIOSI RASP HTTP binding - Sending -->
      <binding name="OiosiHttpEndpointBinding" closeTimeout="00:05:00"
openTimeout="00:05:00" receiveTimeout="00:05:00" sendTimeout="00:05:00">
        <reliableSession flowControlEnabled="false" ordered="true"
maxTransferWindowSize="32" maxPendingChannels="32" />
        <ubiquitousProperties />
        <clientSignatureValidationProofInterceptor />
        <security
messageSecurityVersion="WSSecurity10WSTrustFebruary2005WSSecureConversationFebruar
y2005WSSecurityPolicy11BasicSecurityProfile10" defaultAlgorithmSuite="Default"
authenticationMode="MutualCertificate" requireDerivedKeys="false"
securityHeaderLayout="Strict" includeTimestamp="true"
keyEntropyMode="CombinedEntropy" messageProtectionOrder="SignBeforeEncrypt"
requireSignatureConfirmation="false">
          <localClientSettings maxClockSkew="00:10:00" />
          <localServiceSettings maxClockSkew="00:10:00" />
          <secureConversationBootstrap>
            <localClientSettings maxClockSkew="00:10:00" />
            <localServiceSettings maxClockSkew="00:10:00" />
          </secureConversationBootstrap>
        </security>
        <clientPartyIdentifierHeader />
        <textMessageEncoding messageVersion="Default" writeEncoding="utf-8">
          <readerQuotas maxArrayLength="200000000" />
        </textMessageEncoding>
        <httpTransport manualAddressing="false" maxReceivedMessageSize="10485760"
maxBufferSize="10485760" />
      </binding>
    </customBinding>
  </bindings>
  ...
<system.serviceModel>
```

First there is a basicHttpBinding, that must be named `NHR_lookup`, that is used to perform lookup in NemHandelsRegisteret.

Second, must of theses configuration should not be changed.
Last, there must be a customBinding where the name (‘`OiosiHttpEndpointBinding`’) must match the defined name in section 4.1.

5 Receiving Configuration - App.Config

Each test project has an App.Config application configuration file, and so should every project that uses the RASP Library.

Most of the Windows Communication Foundation settings are done in App.Config, and this section will briefly cover each important section in the configuration.

It is recommended that the main sections described here are copied from one of the test projects, since the RASP Library is dependent on default settings (first and foremost the OiosiHttpServer, OiosiHttpServerBehavior, OiosiHttpServerBinding).

Note – This section describe how to setup a receiving endpoint on an IIS (IIS hosted). There exist other ways of setting up receiving service (self-hosted), that will not be described here.

5.1 Server services

App.Config sample 4

```
<system.serviceModel>
  ...
  <services>
    <!-- The OIOSI RASP HTTP service - Receiving  -->
    <service behaviorConfiguration="OiosiHttpServerBehaviour"
             name="Company.Product.Component.OiosiOmniEndpoint">
     <endpoint name="OiosiHttpServer"
               binding="customBinding"
               bindingConfiguration="OiosiHttpServerBinding"
               contract="dk.gov.oiosi.communication.client.IClientProxyContract" />
    </service>
  </services>
 ...
<system.serviceModel>
```

This sample shows a server endpoint using the attributes

Service

- **behaviorConfiguration** – The name of the behavior to use, described in section 5.2.
- **name** – Is the namespace and class that implement the communication interface.

Services

- **name** – Name of the endpoint – not importen.
- **binding** – Must be 'customBinding', as it described that we used a custom binding.
- **bindingConfiguration** – The name of the binding to use, described in the customConfiguration section 5.3.
- **contact** – Represents service contract implemented by the endpoint, describing what operations are supported and what SOAP actions they expect (and return). RASP endpoints should implement dk.gov.oiosi.communication.client.IClientProxyContract interface.

5.2 Server behavior

The behavior configuration for sending.

App.Config sample 5

```
<system.serviceModel>
   ...
  <behaviors>
      <endpointBehaviors>
        <behavior name="OiosiHttpServerBehavior"
                  returnUnknownExceptionsAsFaults="true">
          <serviceMetadata httpGetEnabled="true" />
          <signCustomHeaders>
            <headers>
              <add name="MessageIdentifier"
namespace="http://rep.oio.dk/oiosi.ehandel.gov.dk/xml/schemas/2007/09/01/" />
              <add name="SenderPartyIdentifier"
namespace="http://rep.oio.dk/oiosi.ehandel.gov.dk/xml/schemas/2007/09/01/" />
              <add name="SenderPartyIdentifierType"
namespace="http://rep.oio.dk/oiosi.ehandel.gov.dk/xml/schemas/2007/09/01/" />
              <add name="ReceiverPartyIdentifier"
namespace="http://rep.oio.dk/oiosi.ehandel.gov.dk/xml/schemas/2007/09/01/" />
              <add name="ReceiverPartyIdentifierType"
namespace="http://rep.oio.dk/oiosi.ehandel.gov.dk/xml/schemas/2007/09/01/" />
            </headers>
          </signCustomHeaders>

           <serviceCredentials>
            <serviceCertificate storeLocation="LocalMachine" storeName="My"
                                x509FindType="FindBySerialNumber"
                                findValue="56 df e9 a7" />
            <clientCertificate>
              <authentication certificateValidationMode="None"
                              revocationMode="NoCheck" />
            </clientCertificate>
          </serviceCredentials>
          <serviceThrottling />
          <serviceAuthorization impersonateCallerForAllOperations="false"
                                principalPermissionMode="None" />
        </behavior>
      </endpointBehaviors>
    </behaviors>
  ...
<system.serviceModel>
```

App.Config sample 6 shows an endpoint behavior. The behavior name (`OiosiHttpServerBehavior`) must the name defined in the 5.1. The behavior add the `signCustomHeaders` and references a client certificate (and where to find it). As mentioned before, this behaviour is referenced from a client endpoint.
The sender certificate and the server root certificate is checked using WCF extension, as this give the possibility to create better error description back to the sender when something is wrong.

Note – Oiosi RASP can use both Test and Live certificates, however most organisation can't use test certificates in testing. This is because of Nets port restrain in there firewall protection the LDAR and OCSP/CRL servers.

For a hint on how to find the serial number and store location/name of your certificate check the section named "*Importing certificates*" in the document "Rasp Library for .Net Installation".

The certificate configuration must match the location where the certificate is installed. In this setup, the find value is the certificate serial number.

5.3 Server binding

The server binding configuration.

App.Config sample 6

```
<system.serviceModel>
   ...
    <bindings>
      <customBinding>
        <binding name="OiosiHttpServerBinding" closeTimeout="00:05:00"
                 openTimeout="00:05:00" receiveTimeout="00:05:00"
                 sendTimeout="00:05:00">
          <serverSchematronValidationInterceptor ValidateRequest="true"
ValidateResponse="false" FaultOnRequestValidationException="true" />
          <serverSchemaValidationInterceptor ValidateRequest="true"
ValidateResponse="false" FaultOnRequestValidationException="true" />
          <serverCertificateValidationInterceptor ValidateRequest="true"
ValidateResponse="false" FaultOnRequestValidationException="true" />
          <serverSignatureValidationProofInterceptor ValidateRequest="true"
ValidateResponse="false" FaultOnRequestValidationException="true" />
          <reliableSession inactivityTimeout="00:05:00" maxRetryCount="8"
ordered="true" />
          <security defaultAlgorithmSuite="Default"
authenticationMode="MutualCertificate" requireDerivedKeys="false"
securityHeaderLayout="Strict" includeTimestamp="true"
messageProtectionOrder="SignBeforeEncrypt"
messageSecurityVersion="WSSecurity10WSTrustFebruary2005WSSecureConversationFeb
ruary2005WSSecurityPolicy11BasicSecurityProfile10"
requireSignatureConfirmation="false">
            <localClientSettings maxClockSkew="00:10:00" />
            <localServiceSettings maxClockSkew="00:10:00" />
            <secureConversationBootstrap>
              <localClientSettings maxClockSkew="00:10:00" />
              <localServiceSettings maxClockSkew="00:10:00" />
            </secureConversationBootstrap>
          </security>
          <serverPartyIdentifierHeader />
          <textMessageEncoding messageVersion="Default" writeEncoding="utf-8">
            <readerQuotas maxArrayLength="2147483647" />
          </textMessageEncoding>
          <httpTransport manualAddressing="false"
maxReceivedMessageSize="2147483647" maxBufferSize="2147483647" />
        </binding>
      </customBinding>
    </bindings>
  ...
<system.serviceModel>
```

The name of the server binding (`OiosiHttpServerBinding`) must match the name of the binding defined in section 5.1.

5.4 Configuring timeouts

There are several timout settings in the app.config. Some of the overall timeouts are described here.

- SendTimeout is the overall timeout for a communication, i.e. including all RM messages back and forth between sender and receiver.
- Open- and CloseTimeout concerns the timeout of the creation of a connection from the sender to the receiver, and nothing else.
- ReceiveTimeout is the timeout for a process waiting for a message within a session, before deciding to time out the session.

6 Lesson: dk.gov.oiosi.communication

The communication namespace holds the 3 main classes for communicating with web services using the RASP stack. These are

- Request: Allows the simplest form of request using the RASP stack using http transport. Transport options (i.e. using RM, security, schema- and schematron validation) can be configured in App.Config as described in the earlier section.

6.1 Request

dk.gov.oiosi.communication.Request is the main class for making RASP service calls.

For a concrete example of how to use the Request class, see the test project dk.gov.oiosi.test.request.

The sample below shows how to easily use Request to send an XML document to an http endpoint through the use of the method *GetResponse*. The code should be fairly straight forward.

Code sample 1

```
//Sends an xml document and receives a response
XmlDocument xdoc = new XmlDocument();
Request raspReq = new Request(new Uri("http://myEndpoint"));
Response response;

try{
      raspReq.GetResponse(new OiosiMessage(xdoc), out response);
}
catch(RequestShutdownException e){
      //No need to do anything in particular if one
      //isn't concerned with a nice shutdown
      // If the response variable is set, it's good to use
      // and your message has been acknowledged
}
```

Calls can be made to http service, and Request automatically detects which type of service is being called by looking at the scheme of the URI given. Http endpoint addresses MUST be formatted *http://address* and is normally provided by NemHandelsRegisteret, and is currently the only one supported.

Code sample 1 give an URI as lone argument to the Rasp constructor, defining what endpoint messages will be sent to. However, Request offers two more constructors, presented in Code sample 2.

Code sample 2

```
//Takes the name of an endpoint in App.Config
public Request(string endpointConfigurationName);

//Takes an endpoint, credentials and a sending policy
public Request(Uri endpointAddress,
               Credentials credentials,
               SendPolicy sendPolicy);
```

The first constructor in Code sample 2 takes the name of an endpoint in the application configuration file. See the file App.Config in the test dk.gov.oiosi.test.request for further reference to how the configured endpoint (OiosiHttpEndpoint) describes the service which we will call.

An endpoint configuration is needed no matter what constructor is used. Unless the first constructor in Code sample 2 is used, the endpoint configuration name must be "**OiosiHttpEndpoint"**, and therefore should always be present in the App.Config file when using the RASP Library. See the App.Config section for more information.

The second constructor in Code sample 2 takes an endpoint URI, just like the ones used in Code sample 1+2, but also takes programmatically set certificates (for sending and/or receiving). These will override any certificates given in App.Config.

6.2 Extended request

This section demonstrates the following in addition to making plain Request:

- Gets identifiers from xml documents (e.g. EAN or OVT numbers) from documents using configurable xpath expressions
- Performs a UDDI lookup using these parameters
- Retrieves an endpoint certificate from LDAP based on information returned from UDDI
- Checks certificate revocation status against OCSP.

Please note that the project has moved into the samples namespace and changed name to "dk.gov.oiosi.samples.consoleClientExample".

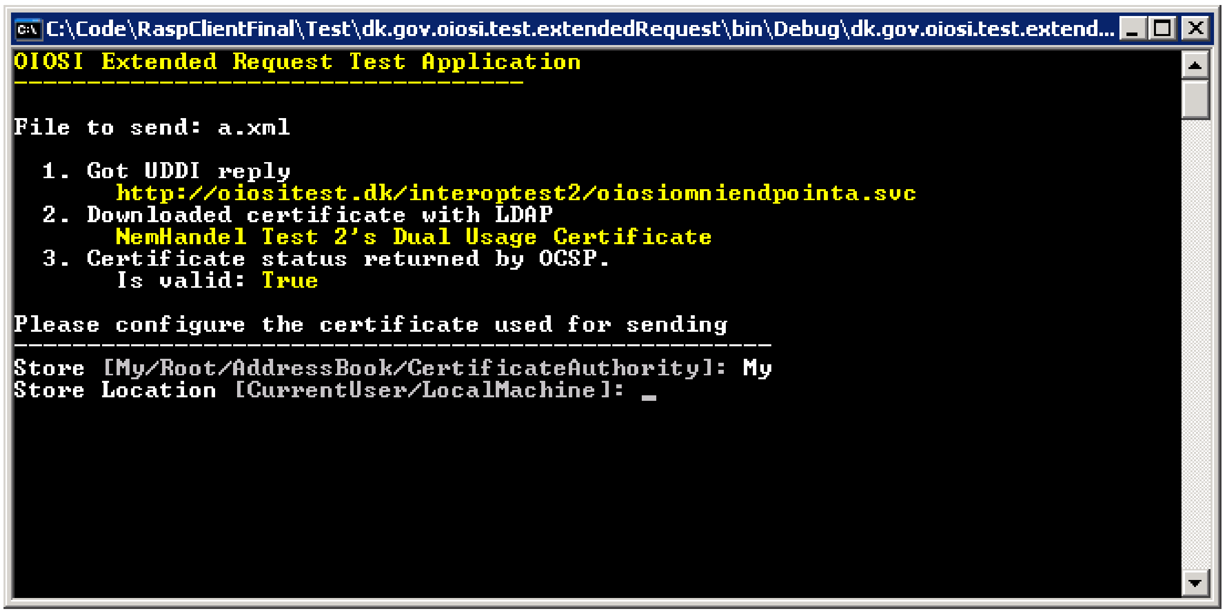
For a concrete example of how to use the Request class, see the test project dk.gov.oiosi.test.requestTests.

Code sample 3 shows how ExtendedRequest encapsulates all UDDI, OCSP, LDAP and document searching to send an XML document to an http endpoint through the use of the method *GetResponse*.

Code sample 3

```
// 1. Get client certificate:
X509Certificate2 cert = CertificateLoader.GetCertificateFromStoreWithSSN(
       "CVR:26769388-UID:1172691221366",
       StoreLocation.CurrentUser,
       StoreName.My
);

OcesX509Certificate clientCert = new OcesX509Certificate(cert);

// 2. Define send policy:
SendPolicy sendPolicy = new SendPolicy("*");

// 3. Create request:
ExtendedRequest requestEx = new ExtendedRequest(clientCert, sendPolicy);

// 4. Create test message:
// 4.1 Load a test message from file:
XmlDocument xmlMsg = GetTestMessage();
OiosiMessage msg = new OiosiMessage(xmlMsg);

// 5. Get response:
Response response;
try {
       response = requestEx.GetResponse(msg);
} catch (Exception ex) {
       txtResult.Text += "RequestExtended failed: " + ex.ToString();
       return;
}
```

You can run the ExtendedRequest test sample by running the dk.gov.oiosi.test.extendedRequest project application, see below.



```
C:\Code\RaspClientFinal\Test\dk.gov.oiosi.test.extendedRequest\bin\Debug\dk.gov.oiosi.test.extend...
OIOSI Extended Request Test Application
-----------------------------------

File to send: a.xml

  1. Got UDDI reply
       http://oiositest.dk/interoptest2/oiosiomniendpointa.svc
  2. Downloaded certificate with LDAP
       NemHandel Test 2's Dual Usage Certificate
  3. Certificate status returned by OCSP.
       Is valid: True

Please configure the certificate used for sending
---------------------------------------------------
Store [My/Root/AddressBook/CertificateAuthority]: My
Store Location [CurrentUser/LocalMachine]: _
```

The LDAP, OCSP and UDDI lookup components have both online and offline implementation of their interfaces, see the RASP client tutorials on how to switch between these.

You can choose which implementation to run by setting the corresponding factory configuration in the configuration file, see below.

You choose the implementation type by setting class + namespace + assembly name. The factory then instantiates this type dynamically. You may supply your own implementation of the OCSP, LDAP and UDDI interfaces.

Code sample 4

```
<ConfigurationSection xsi:type="RevocationLookupFactoryConfig">
    <ImplementationNamespaceClass>
             dk.gov.oiosi.security.revocation.ocsp.OcspLookup
    </ImplementationNamespaceClass>
    <ImplementationAssembly>dk.gov.oiosi.library</ImplementationAssembly>
</ConfigurationSection>
<ConfigurationSection xsi:type="LdapLookupFactoryConfig">
       <ImplementationNamespaceClass>
             dk.gov.oiosi.security.Ldap.LdapCertificateLookup
       </ImplementationNamespaceClass>
       <ImplementationAssembly>
             dk.gov.oiosi.library
       </ImplementationAssembly>
</ConfigurationSection>
<ConfigurationSection xsi:type="UddiLookupClientFactoryConfig">
       <ImplementationNamespaceClass>
             dk.gov.oiosi.uddi.UddiLookupClient
       </ImplementationNamespaceClass>
       <ImplementationAssembly>
             dk.gov.oiosi.library
       </ImplementationAssembly>
</ConfigurationSection>
```

Code sample 5 uses OCSP for to check for revocation, while sample 6 use CRL lookup.

Code sample 5

```
<ConfigurationSection xsi:type="RevocationLookupFactoryConfig">
    <ImplementationNamespaceClass>
       dk.gov.oiosi.security.revocation.crl.CrlLookup
     </ImplementationNamespaceClass>
    <ImplementationAssembly>dk.gov.oiosi.library</ImplementationAssembly>
</ConfigurationSection>
<ConfigurationSection xsi:type="LdapLookupFactoryConfig">
       <ImplementationNamespaceClass>
             dk.gov.oiosi.security.Ldap.LdapCertificateLookup
       </ImplementationNamespaceClass>
       <ImplementationAssembly>
             dk.gov.oiosi.library
       </ImplementationAssembly>
</ConfigurationSection>
<ConfigurationSection xsi:type="UddiLookupClientFactoryConfig">
       <ImplementationNamespaceClass>
             dk.gov.oiosi.uddi.UddiLookupClient
       </ImplementationNamespaceClass>
       <ImplementationAssembly>
             dk.gov.oiosi.library
       </ImplementationAssembly>
</ConfigurationSection>
```

The OCSP-server is by default read from the certificate. This can be overridden by having a configuration in the OCSPconfig section:

Code sample 6

```
<ConfigurationSection xsi:type="OcspConfig">
    <DefaultTimeoutMsec>10000</DefaultTimeoutMsec>
    <ServerUrl>http://localhost/</ServerUrl>
</ConfigurationSection>
```

The configuration file also demonstrates how to configure RaspDocumentType configuration. This configuration is associates the following information with the root element of an xml document:

- Validation schemas and schematron xslts
- Xpath expressions for finding e.g. an EAN number or other endpoint key types
- Associate a service and SOAP action with the document type

Most of this configuration points forward to the RASP client which uses this library for automatically sending business messages.

You can see the RequestExtended class for an example on how to string the OCSP, LDAP, UDDI, certificate checking and Request components together, either by using configuration or programmatically.

6.2.1 The UDDI lookup

The UDDI lookup of the extended request requires the UDDI connection to be configured in the app.config file, where it has its own HTTP binding.

The parameters of the UDDI lookup may additionally use a process definition filter. By setting the BusinessProcessDefinitionTModel and RoleIdentifier fields of the LookupParameters object, the result from the query is filtered using these criteria. If null, they are ignored.

For more information on how to use the IUddiLookup interface have a look at section 7.

7 Lesson: dk.gov.oiosi.uddi.IUddiLookupClien

This section will have a short description on how to use the UddiLookupClient interface and the LookupParameters class for different scenarios.

There are several different kinds of lookup in that the interface IUddiLookupClient supports. The interface only has one method and that is for lookup that take some parameters and returns a list of responses. Se

```
/// <summary>
/// Translate interface for the ARS (Address Resolving Service) client.
/// </summary>
public interface IUddiLookupClient {
	/// <summary>
	/// Translate parametres
	/// </summary>
	/// <param name="parameters"></param>
	/// <returns></returns>
	List<UddiLookupResponse> Lookup(LookupParameters parameters);
}
```

The parameters that are used for the lookup can be different after what is searched for in the UDDI. It seems like there are three major lookups that is needed by suppliers and these are; first one is to find all that a specific identifier supports, second one is to find the endpoint to a specific identifier and specific document type, finally the third one is to find the endpoint to a specific identifier, specific document type and specific profile.

First there is how to make a lookup that get all services for a specific identifier, then you have to use the following constructor:

```
public LookupParameters(
	Identifier identifier,
	List<EndpointAddressTypeCode> acceptedTransportProtocols)
```

Where the identifier is given as the first parameter and the second parameter is what transport protocols the client can support.

Second lookup type takes three parameters; the first parameter is the identifier, the second parameter is the service identifier in the UDDI and the third parameter is the accepted transport protocols. The second parameter is a bit tricky to find but it is defined and found on the UDDI. Se below for the method parameters:

```
public LookupParameters(
	Identifier identifier,
	UddiId serviceId,
	List<EndpointAddressTypeCode> acceptedTransportProtocols)
```

Here the UDDIID on the portType tModel is used as the value in the parameter. The portType can for an example correlate to a document type (invoice) in a process (billing).

Third lookup type takes four parameters where the third parameter is different than the other two lookup constructors. This parameter is a list of UddiId's on the profiles that the service must support. The method looks like the following:

```
public LookupParameters(
	Identifier identifier,
	UddiId serviceId,
      List<UddiId> profileIds,
      List<EndpointAddressTypeCode> acceptedTransportProtocols)
```
The lookup will accept a service as a result if just one of the profiles in the list is supported by it.

There are more constructors but they are not needed to send documents over the RASP protocol, so they are not described here in this document.

8 Lesson: dk.gov.oiosi.extension.wcf

The RASP library comes with several extensions to the .Net 3 Windows Communications Foundation framework, that can all be found under the dk.gov.oiosi.extension.wcf and dk.gov.oiosi.raspProfile.extension.wcf namespaces.

These extensions come in the form of binding elements that are inserted into the communication stack, where they intercept and handle the in- or outgoing message according to their functionality.

The extensions available in version 2.0 are

Interceptors

- The schema interceptor
- The schematron interceptor
- The signature validation proof generator
- The XSLT transformer
- The ubiquitous message property interceptor

Headers

- The party identifier headers

These stack elements are added in the App.Config file, but only after adding reference to each of the elements configuration extension as seen in App.Config sample 7.

Server and client side interceptors have been implemented in different manners because of the different ways the two handles SOAP messages wherefore one has to make sure the correct interceptor binding element has been selected.

App.Config sample 7

```
<!-- Our binding extension, letting WCF know where our custom WCF components are
implemented -->
  <extensions>
    <bindingElementExtensions>

      <!-- Signature validation proof generation -->
      <add name="serverSignatureValidationProofInterceptor"
      type="dk.gov.oiosi.extension.wcf.Interceptor.Security.ServerSignatureValidatio
      nProofBindingExtensionElement, dk.gov.oiosi.library" />
      <add name="clientSignatureValidationProofInterceptor"
      type="dk.gov.oiosi.extension.wcf.Interceptor.Security.ClientSignatureValidatio
      nProofBindingExtensionElement, dk.gov.oiosi.library" />

      <!-- Schema validation -->
      <add name="serverSchemaValidationInterceptor"
      type="dk.gov.oiosi.extension.wcf.Interceptor.Validation.Schema.ServerSchemaVal
      idationBindingExtensionElement, dk.gov.oiosi.library" />
      <add name="clientSchemaValidationInterceptor"
      type="dk.gov.oiosi.extension.wcf.Interceptor.Validation.Schema.ClientSchemaVal
      idationBindingExtensionElement, dk.gov.oiosi.library" />

      <!-- Schema validation -->
      <add name="serverSchematronValidationInterceptor"
      type="dk.gov.oiosi.extension.wcf.Interceptor.Validation.Schematron.ServerSchem
      atronValidationBindingExtensionElement, dk.gov.oiosi.library" />
      <add name="clientSchematronValidationInterceptor"
      type="dk.gov.oiosi.extension.wcf.Interceptor.Validation.Schematron.ClientSchem
      atronValidationBindingExtensionElement, dk.gov.oiosi.library" />

      <!-- Custom RASP headers-->
      <add name="clientPartyIdentifierHeader"
      type="dk.gov.oiosi.raspProfile.extension.wcf.Interceptor.CustomHeader.ClientPa
      rtyIdentifierHeaderBindingExtensionElement, dk.gov.oiosi.raspProfile" />
      <add name="serverPartyIdentifierHeader"
      type="dk.gov.oiosi.raspProfile.extension.wcf.Interceptor.CustomHeader.ServerPa
      rtyIdentifierHeaderBindingExtensionElement, dk.gov.oiosi.raspProfile" />

      <!-- Adds parameters to ALL messages, including RM messages -->
      <add name="ubiquitousProperties"
      type="dk.gov.oiosi.extension.wcf.Interceptor.UbiquitousProperties.UbiquitousPr
      opertiesBindingExtensionElement, dk.gov.oiosi.library" />
    </bindingElementExtensions>

      <behaviorExtensions>
            <!-- Behavior that selects headers to be added for signing -->
            <add name="signCustomHeaders"
            type="dk.gov.oiosi.extension.wcf.Behavior.SignCustomHeadersBehaviorExten
            sionElement, dk.gov.oiosi.library, Version=2.1.0.0, Culture=neutral,
            PublicKeyToken=null" />
      </behaviorExtensions>

</extensions>
```

8.1 Schema and schematron interceptors

The xml validators have the following settings

- **ValidateRequest** - If true it will validate the xml on request. Default is true.
- **ValidateResponse** - If true it will validate the xml on response. Default is true. The current configuration does not return any valid xml so this is disabled.
- **FaultOnRequestValidationException** - If true it will send soap fault to the client if the validation fails. If false the message will continue up the stack and any validation failure is added as a custom property to the message. Default is true.

8.2 Signature validation proof interceptor

The server side signature validation proof interceptor has the following options

- **FaultOnRequestValidationException** - should a SOAP fault be sent on exceptions? Default is true.

The client side signature validation proof interceptor has no options.

Note that the ServerSignatureProof and ClientSignatureProof interceptors must be located between the RM layer and the security layer as seen in **Error! Reference source not found.**.

8.3 XSLT Transformation interceptor

The XSLT transformation transforms the incoming XML, and has the following options

- **FaultOnTransformationException –** should a SOAP fault be sent on exceptions?
- **PropagateOriginalMessage** – The original XML will be added as a message property

Note that the XSLT interceptor must be placed above the ReliableMessaging layer.

8.4 Ubiquitous properties interceptor

The ubiquitous properties interceptor adds ubiquitous message properties to all messages that pass it (as opposed to normal WCF Message properties), that will only be added to the payload message.

Ubiquitous properties are added to the OiosiMessage before sending, and need to be given a unique string as an identifier, which later stack layers need to be familiar with if they would like to read the property.

```
OiosiMessage msg = new OiosiMessage();
msg.UbiquitousProperties.Add("MyProperty", new object());
```

Note that the ubiquitous properties interceptor needs to be located under the ReliableMessaging layer.

8.5 SenderPartyIdentifier and ReceiverPartyIdentifier headers

In dk.gov.oiosi.raspProfile.communication.extension.wcf an additional interceptor can be found, which is used to add the obligatory RASP SOAP headers <SenderPartyIdentifier> and <ReceiverPartyIdentifier> .

The value of the headers is configured by adding an PartyIdentifierSettings object as an ubiquitous property to the message to be sent, as seen in the code below. The name of the ubiquitous property must be the value found in the constant MessagePropertyKey on the PartyIdentifierHeaderSettings class.

```
OiosiMessage msg = new OiosiMessage();

string key = PartyIdentifierHeaderSettings.MessagePropertyKey;
PartyIdentifierHeaderSettings partyIdentifierSetting = new
PartyIdentifierHeaderSettings(senderID, receiverID);

msg.UbiquitousProperties[key] = partyIdentifierSetting;
```

9 Lesson: Enabling tracing

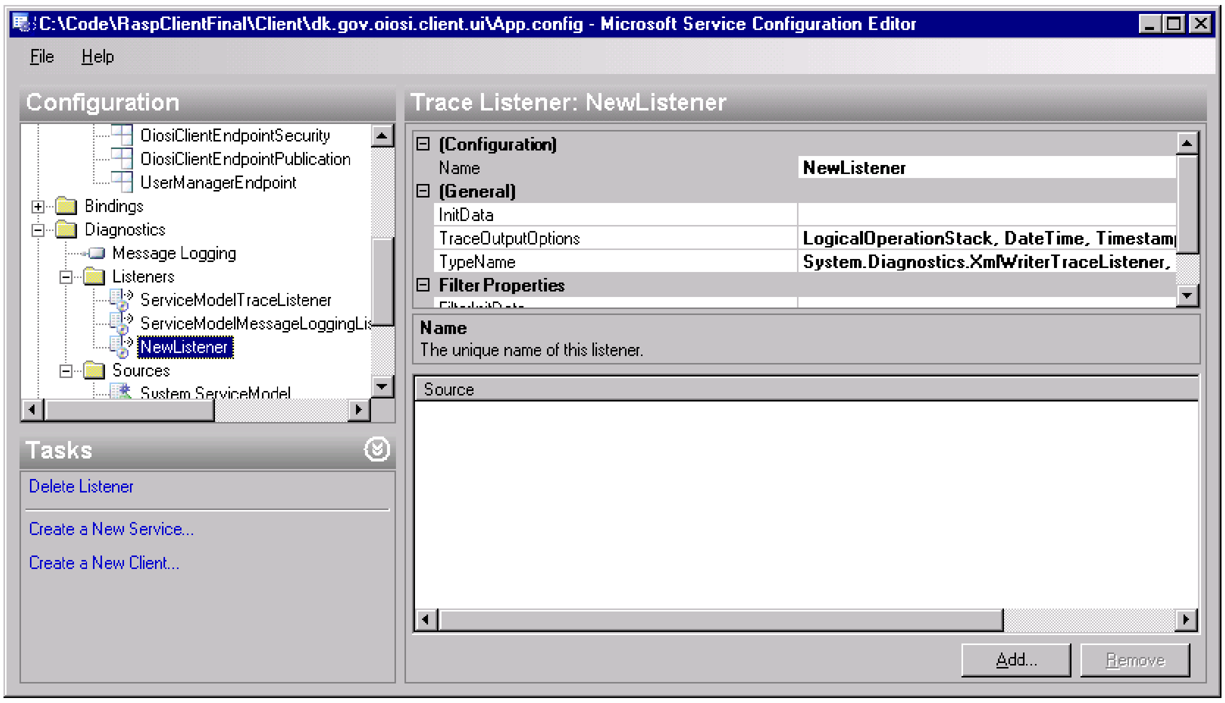
Windows communication enables both Message level logging and internal system traces, both which can be used with the RASP library. Furthermore the RASP library can add it's own internal system logs to the same (or a different) log file as WCF. Beware that when using traces for the library on the Verbose tracing level the trace files will very quickly become very large, so tracing at this level should only be enabled for advanced debugging.

At the very top of all the App.Config files that are distributed with the RASP library source, there should be a section called `<system.diagnostics>.` This section has been disabled by making it an xml comment, and to enable it again just remove the <!-- and .--> from before and after it.

When tracing has been enabled, two files (App_Traces.svclog and App_Messages.svclog) can be viewed using the application SvcTraceViewer.exe that comes with the Windows SDK. App_Traces will contain internal logs and App_Messages will contain all the SOAP messages sent and/or received.

9.1 Changing the trace settings

To change the tracing options it is recommended that the App.Config file is edited in the Windows SDK application SvcConfigEditor.exe.


C:\Code\RaspClientFinal\Client\dk.gov.oiosi.client.ui\App.config - Microsoft Service Configuration Editor
File
Help
Configuration
OiosiClientEndpointSecurity
OiosiClientEndpointPublication
UserManagerEndpoint
Bindings
Diagnostics
Message Logging
Listeners
ServiceModelTraceListener
ServiceModelMessageLoggingLis
NewListener
Sources
System.ServiceModel
Tasks
Delete Listener
Create a New Service...
Create a New Client...
Trace Listener: NewListener
Source
Add...
Remove

(Configuration)
Name NewListener
(General)
InitData
TraceOutputOptions LogicalOperationStack, DateTime, Timestam
TypeName System.Diagnostics.XmlWriterTraceListener,
Filter Properties

Name
The unique name of this listener.

In the configuration editor, under Diagnostics there will be two important sections; Listeners and Sources.

Under listeners you can add new trace listeners that will write to other files than the above mentioned two. Under Sources you can add more sources (for example your own WCF extensions) or change to what Listener the existing Sources will be written (if for example you would like to separate RASP logs from WCF logs). It is also here where you can change on what detail level to log. “Warning” is recommended to not clog the log files.

10 Lesson: Configuration – RaspConfiguration.xml

This section briefly describes how to access the dynamic configuration file, which supplements App.Config, RaspConfiguration.xml and change loaded library versions from live to test versions.

The RaspConfiguration.xml file is loaded using the class `ConfigurationDocument`. Default the location and name of the configuration file is loaded from the programs main App.Config file, or it should be set like this:

```
        ConfigurationDocument.ConfigFilePath = "RaspConfiguration.Live.xml";
```

Note – You must set the configuration before starting to use the RASP functionality.

You can choose to use offline test stub versions of the LDAP, OCSP and UDDI libraries. This is suitable for testing in offline environments or to fix some parameters of the test.

You can set this in the factory configuration sections of the config.

To do use test stubs do the following:

- **LDAPLookupFactory –** change the implementation namespace class from “dk.gov.oiosi.security.Ldap.LdapCertificateLookup” to “dk.gov.oiosi.security.Ldap.LdapCertificateLookupTest”.
- **LdapLookupFactoryConfig –** change the implementation namespace class from “dk.gov.oiosi.security.Ldap.LdapCertificateLookup” to “dk.gov.oiosi.security.Ldap.LdapCertificateLookupTest”.
- **RevocationLookupFactoryConfig –** change the implementation namespace class from “dk.gov.oiosi.security.revocation.ocsp.OcspLookup” to “dk.gov.oiosi.security.revocation.ocsp.OcspLookupTest”.

When you use the test stubs, you can configure the behaviour of each of them. You can do this in the following sections:

- LdapCertificateLookupTestConfig: Here you can configure a certificate that the LDAP client should always return.
- OcspLookupTestConfig: Here you can set the response that the OCSP client always will return in response to a question of certificate validity (true/false).
- UddiLookupClientTestConfig: Here you can statically configure an UDDI response, regardless of lookup parameters. Parameters include the endpoint address and certificate subject.

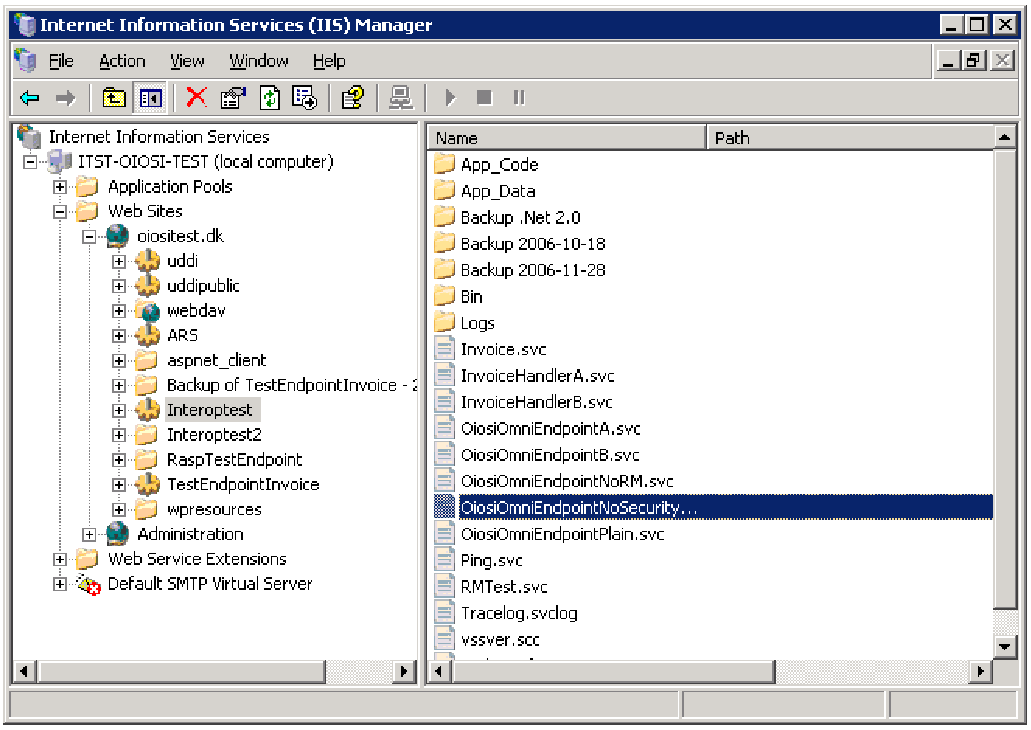
11 Lesson: Setting up an IIS hosted RASP service

To host an HTTP endpoint it is recommended that this endpoint should be hosted by Microsoft Internet Information Services (IIS).

A test project in the sample code demonstrates how this could be done. In this project you will find some code and a Web.Config file (which will act as a substitute for the App.Config file while hosting our service in IIS).

To get started, either publish the project directly into *C:\Inetpub\wwwroot\* (assuming that you have your Windows installation on the C drive), or publish another location and manually install the application into the IIS.

Then you should open the IIS manager (found under Start->Control Panel->Administrative Tools, alternatively as a subsection of Start->Control Panel->Administrative Tools->Computer Management).

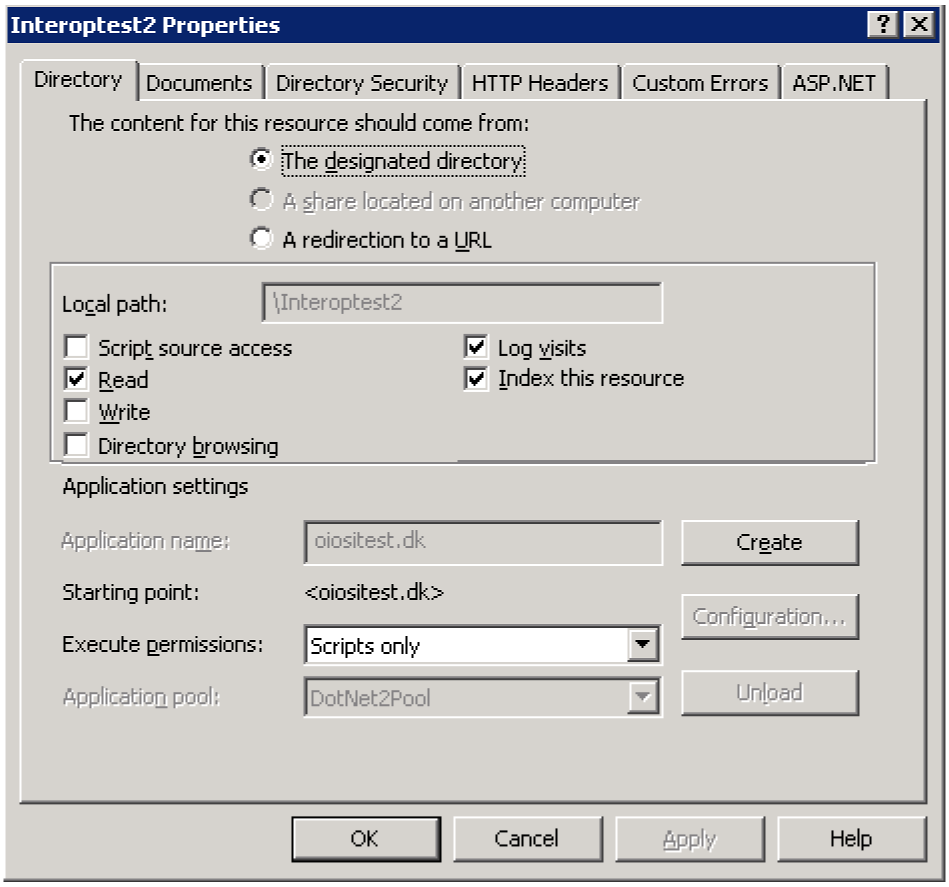


```
Internet Information Services
  ITST-OIOSI-TEST (local computer)
    Application Pools
    Web Sites
      oiositest.dk
        uddi
        uddipublic
        webdav
        ARS
        aspnet_client
        Backup of TestEndpointInvoice - 2
        Interoptest
        Interoptest2
        RaspTestEndpoint
        TestEndpointInvoice
        wpresources
      Administration
    Web Service Extensions
    Default SMTP Virtual Server
```

```
App_Code
App_Data
Backup .Net 2.0
Backup 2006-10-18
Backup 2006-11-28
Bin
Logs
Invoice.svc
InvoiceHandlerA.svc
InvoiceHandlerB.svc
OiosiOmniEndpointA.svc
OiosiOmniEndpointB.svc
OiosiOmniEndpointNoRM.svc
OiosiOmniEndpointNoSecurity...
OiosiOmniEndpointPlain.svc
Ping.svc
RMTest.svc
Tracelog.svclog
vssver.scc
```

Under local computer\Web sites there should be a default web site with the folder *C:\Inetpub\wwwroot\* as it's home directory. If there is no web sites please refer to Microsoft help for setting a web site up.

As a sub folder of this web site you should see the folder you just published called *RaspTestEndpoint*. Right click on this folder and select *Properties*.


Interoptest2 Properties

The content for this resource should come from:

[✓ The designated directory]

A share located on another computer

A redirection to a URL

Local path: \Interoptest2

Script source access

[✓ Read]

Write

Directory browsing

[✓ Log visits]

[✓ Index this resource]

Application settings

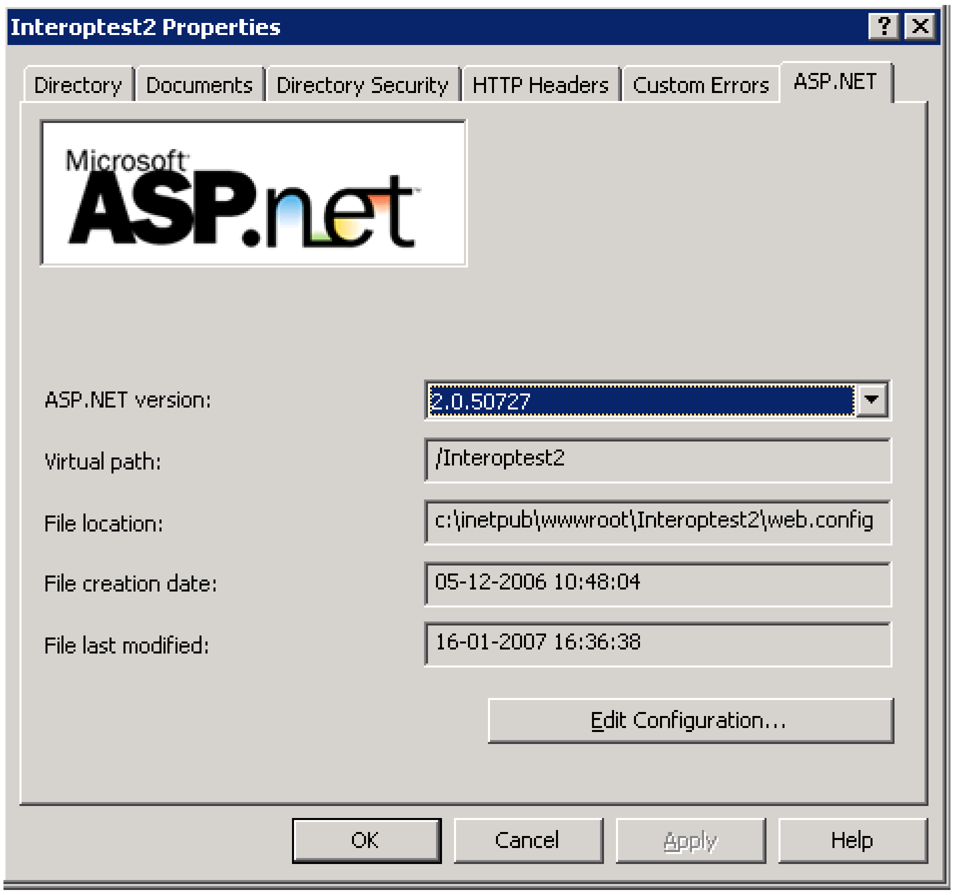
Application name: oiositest.dk

Starting point: <oiositest.dk>

Execute permissions: Scripts only

Application pool: DotNet2Pool

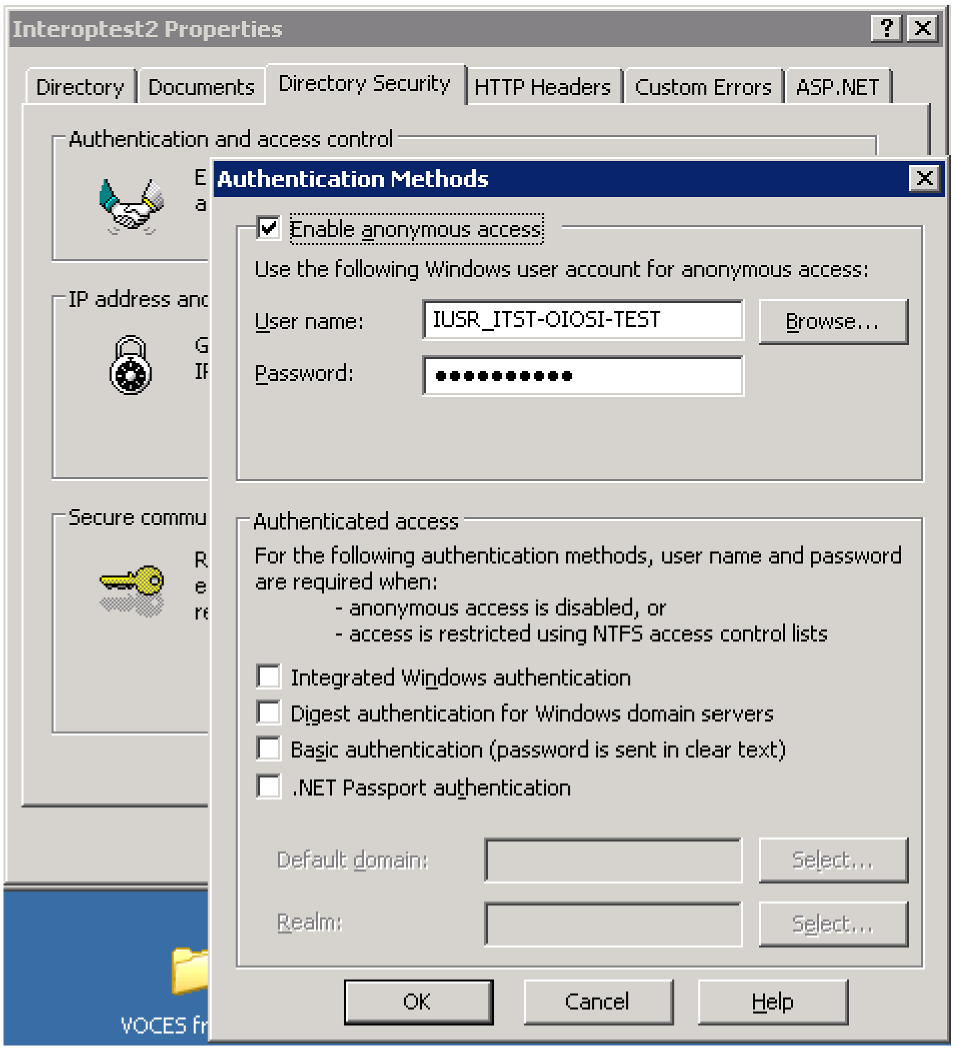
In the properties window, under the *Directory* tab push the *Create* button.


Interoptest2 Properties

Directory | Documents | Directory Security | HTTP Headers | Custom Errors | ASP.NET

ASP.NET version:	2.0.50727
Virtual path:	/Interoptest2
File location:	c:\inetpub\wwwroot\Interoptest2\web.config
File creation date:	05-12-2006 10:48:04
File last modified:	16-01-2007 16:36:38

In the properties window, under the *ASP.NET* tab, select Asp.Net version 2.0


Interoptest2 Properties

Directory | Documents | Directory Security | HTTP Headers | Custom Errors | ASP.NET

Authentication and access control

Authentication Methods

[✓] Enable anonymous access

Use the following Windows user account for anonymous access:

User name: IUSR_ITST-OIOSI-TEST

Password: ●●●●●●●●●●●

Authenticated access

For the following authentication methods, user name and password are required when:
- anonymous access is disabled, or
- access is restricted using NTFS access control lists

[ ] Integrated Windows authentication
[ ] Digest authentication for Windows domain servers
[ ] Basic authentication (password is sent in clear text)
[ ] .NET Passport authentication

Default domain:

Realm:

Under the *Directory Security* tab, *Authentication and access control,* make sure anonymous access is enabled.

Now you should be able to see that your service is running by opening

http://localhost/RaspTestEndpoint/OiosiOmniEndpoint.svc

in a browser (such as Internet Explorer) .

11.1.1 Tips for solving common issues when hosting in IIS

If you have problems contacting the ISS service from outside, you may try and look into the firewall settings.

If your endpoint is not working, you may try the following:

- Go to the IIS application pool property window
- Select the "Identity" tab.
- Change the account to "local system"

If you do not want to elevate permission on the whole of application pool, you can try the following.

- Creating a new application pool, where all web sites are running on the same .Net version (v. 2.0)

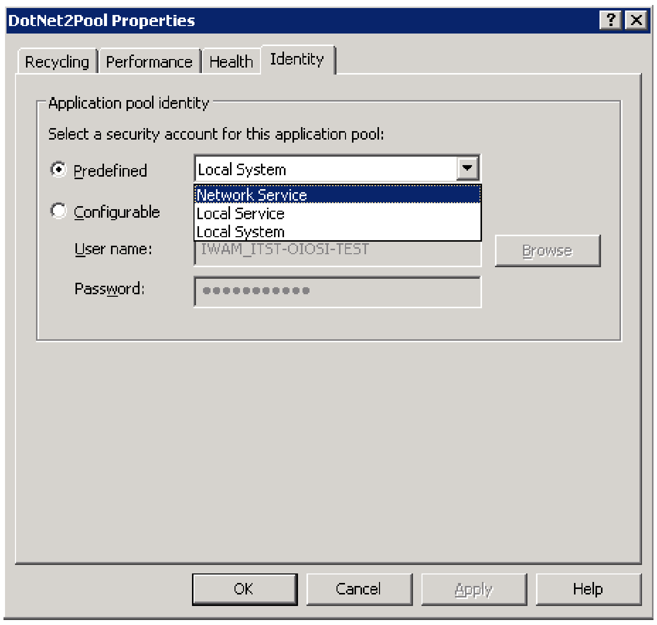
- If that doesn't work, perhaps the application pool doesn't have rights to use the certificate. Try:
    - Delete the certificate OiosiTestVOCES.pfx from the MMC window.
    - Run the command line tool WinHttpCertCfg (http://www.microsoft.com/downloads/details.aspx?familyid=c42e27ac-3409-40e9-8667-c748e422833f&displaylang=en)
    - Re-install the certificate like:

      ```
      >WinHttpCertCfg.exe -i NemHandelTest2.pfx -c LOCAL_MACHINE\MY -a
      "NetworkService" -p Test1234
      ```

      (imports the pfx file to the personal store on local machine, for the NetworkService account, using the password Test1234)
    - Grant acces to the certificate for asp.net by running

      ```
      >WinHttpCertCfg.exe -c LOCAL_MACHINE\MY -s "NemHandel Test 2" -g
      -a "aspnet"
      ```

      (where "*NemHandel Test 2*" is part of the subject string of the certificate you just imported)
    - In web.config, change the location of the certificate from "Root" to "My" store.


DotNet2Pool Properties

Recycling | Performance | Health | Identity

Application pool identity

Select a security account for this application pool:

[✓ Predefined] Local System

Network Service
Local Service
Local System

Configurable

User name: IWAM_ITST-OIOSI-TEST

Password: ●●●●●●●●●●●●

- In the IIS manager, right click on the application pool you're running on (if it is the default, you might want to create a new one) and make sure that under the Identity tab the "Network service" security account is selected.