OIO Service Oriented Infrastructure

OIOSI RASP Library for .NET RC1

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 them used as parts of other components

Please visit http://www.oio.dk/arkitektur/soa/infrastruktur for more information on the OIOSI project.

2 Overview of Lessons

The basics of the RASP Library will be explained through 6 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 lookups)
- Receiving documents
- The interceptors
- Hosting a HTTP service in IIS

3 Prerequisites

3.1 Sample Documents

There are several sample documents used in the tests of the rasp library. They are located in resources directory in the NUnitTestRaspLibrary project file in the RaspClientTests solution.

There are three types of sample documents; xml examples, xml schemas and schematron stylesheets. They are located in the xml, schema and schematron directories respectively.

The xml file “COMPAY_01_01_00_Invoice_v2p0.xml” is an example invoice xml document, the schema file “UBL-Invoice-2.0.xsd” is the invoice schema and the schematron stylesheet “OIOUBL_Invoice_Schematron.xsl” is the invoice schematron stylesheet. The invoice xml file is schema validated by the invoice schema and schematron validated be the invoice schematron stylesheet. The following document types are represented as xml example, xml schema and schematron stylesheet; ApplicationResponse, CreditNote, Invoice, Order, OrderResponseSimple and Reminder.

4 Lesson: 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 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, OiosiEmailEndpoint, OiosiHttpBinding and OiosiEmailBinding).

App.Config sample 1

```
<system.serviceModel>

    <client>
      <!-- The HTTP endpoint-->
      <endpoint name="OiosiHttpEndpoint"
      address="http://... "
      binding="customBinding"
      bindingConfiguration="OiosiHttpBinding"
      contract="Oiosi.Rasp.Communication.Proxy.IRaspProxyContract"
      behaviorConfiguration="OiosiEndpointCertificateBehavior">
        <identity>
          <dns value=" IT- og Telestyrelsen - OIOSI test
            virksomhedssignatur" />
        </identity>
      </endpoint>
    </client>
```

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

- **Address** sets the remote endpoint address, to which requests will be sent
- **Binding/bindingConfiguration** sets the binding of this particular endpoint. In the binding (which is always a customBinding as App.Config sample 1 shows) we will describe the stack setup of the endpoint. More on this in later sections.
- **Contract** represents service contract implemented by the endpoint, describing what operations are supported and what SOAP actions they expect (and return). RASP endpoints should implement Oiosi.Rasp.Communication.Proxy.IRaspProxyContract
- **behaviorConfiguration** defines the endpoint behaviour. The RASP Library uses behaviours to select certificates for sending and receiving.
- **Dns identity** overrides the “dns identity”, to allow clients to communicate with endpoints whose server certificates have domains that do not match the actual server DNS domain.

App.Config sample 2

```
    <behaviors>
      <endpointBehaviors>
        <behavior name="OiosiEndpointCertificateBehavior">
          <clientCredentials>
            <clientCertificate storeLocation="CurrentUser"
                               storeName="My"
                               x509FindType="FindBySerialNumber"
                               findValue=" 40 36 9f 5e"/>
            <serviceCertificate>
              <authentication revocationMode="NoCheck"/>
              <defaultCertificate  storeLocation="LocalMachine"
                                   storeName="Root"
                                   x509FindType="FindBySerialNumber"
                                   findValue="40 36 9f 5e" />
            </serviceCertificate>
          </clientCredentials>
        </behavior>
      </endpointBehaviors>
    </behaviors>
```

App.Config sample 2 shows an endpoint behaviour. The behaviour references a client and a service certificate, and where to find them. As mentioned before, this behaviour is referenced from a client endpoint.

App.Config sample 3

```
    <services>
      <service name="Oiosi.Rasp.Communication.Proxy.RaspService"
      behaviorConfiguration="OiosiServiceCertificateBehavior">

        <endpoint contract="...IRaspServiceContract"
                  address="mailto:server@oiositest.dk"
                  binding="customBinding"
                  bindingConfiguration="OiosiEmailBinding"
                  >
          <identity>
            <dns value="IT- og Telestyrelsen - OIOSI test
                  virksomhedssignatur " />
          </identity>

        </endpoint>
      </service>
    </services>
```

App.Config sample 3 shows a service. Just like the client endpoint, the service has a behaviour, describing what certificate it uses (only this time it is a <serviceBehavior>) and a dns identity. Furhtermore it defines an endpoint, in very much the same way as the client endpoint was defined. **The endpoint address should always be a mail address**, formatted mailto:a@b.com though, since the RASP Library only supports hosting email services. Also, most likely **the service and client endpoint should have different (but similar) binding**, so that mail account settings don't collide (making the server snatching the client's mails and the other way around).

App.Config sample 4

```
<bindings>
   <customBinding>

        <!-- The OIOSI RASP mail binding -->
        <binding  name="OiosiEmailBinding"
                  closeTimeout="00:01:30"
                  openTimeout="00:01:30"
                  receiveTimeout="00:01:30"
                  sendTimeout="00:01:30">

          <reliableSession/>

          <security
          messageSecurityVersion="WSSecurity10WSTrustFebruary2005WSSe
          cureConversationFebruary2005WSSecurityPolicy11BasicSecurity
          Profile10"

                    defaultAlgorithmSuite="Default"
                    authenticationMode="MutualCertificate"
                    requireDerivedKeys="false"
                    securityHeaderLayout="Strict"
                    includeTimestamp="true"
                    keyEntropyMode="CombinedEntropy"
                    messageProtectionOrder="SignBeforeEncrypt"
                    requireSignatureConfirmation="false"

           />

          <textMessageEncoding messageVersion="Default"
                writeEncoding="utf-8"/>

           <OiosiEmailTransport
                outboxImplementation="Oiosi.Rasp.LesnikowskiImplement
                ation.SmtpOutboxLesnikowski,
                Oiosi.Rasp.LesnikowskiImplementation"

                inboxImplementation="Oiosi.Rasp.LesnikowskiImplementa
                tion.Pop3InboxLesnikowski,
                Oiosi.Rasp.LesnikowskiImplementation"

               sendingServerAddress="oiositest.dk"
               receivingServerAddress="oiositest.dk"
               receivingUserName="client"
               receivingPassword="password"
                replyAddress="mailto:client@oiositest.dk"/>
      </binding>
   </customBinding>
</bindings>

<extensions>
   <bindingElementExtensions>
      <add  name="OiosiEmailTransport"
            type="Oiosi.Rasp.WcfExtensions.EmailTransport.RaspEmailBi
            ndingExtensionElement, RaspLibrary"/>
   </bindingElementExtensions>
</extensions>
```

App.Config sample 4 shows an OiosiEmailBinding (the http binding looks the same, only with a httpTransport instead of an OiosiEmailTransport at the end). The binding defines the Windows Communications (WCF) stack that will be used, in the standard RASP case the binding should look as in App.Config sample 4 with a stack looking like

[Application layer]
[Reliable Messaging]
[Security]
[Encoding]
[Transport (http or mail)]

All layers are standard WCF elements, except for the OiosiEmailTransport shown in App.Config sample 4. **To be able to use it, we first need to add the extension** at the end of the code sample, telling WCF where to find the implementation.

The OIOSI Email binding has the following settings

- **outboxImplementation** – the "namespace.name, assembly" of a class that inherits the virtual baseclass Outbox. This implementation handles sending mails. The RASP Library comes with a standard implementation, using the Lesnikowski mail library, as shown in App.Config sample 4.
- **inboxImplementation** – same as above, only implementing Inbox and handling receiving of mails
- **sendingServerAddress/sendingUserName/sendingPassword –** login info for the mailserver used for sending mails. *Only one service or client should use the same binding at a time, to make sure there are no collisions.*
- **receivingServerAddress/receivingUserName/receivingPassword** - login info for the mailserver used for receiving mails *Only one service or client should use the same binding at a time, to make sure there are no collisions.*
- **replyAddress –** the address to which the receiver of messages should reply. In other words, the mailbox that has username *receivingUserName* on server *receivingServerAddress.*
- **pollingInterval –** in seconds, describing how often to poll the mail server when waiting for incoming mails

5 Lesson: Oiosi.Rasp.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 either the email or http transport. Transport options (i.e. using RM, security, schema- and schematron validation) can be configured independently.
- Listener: Server-side functionality using the email transport.
- RequestExtended: The RequestExtended request demonstrates the following in addition to the 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.

5.1 Request

Oiosi.Rasp.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 Test_RaspRequest.

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 = raspReq.GetResponse(new RaspMessage(xdoc));
```

Calls can be made to both http and mail services, 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 mail addresses *mailto:address@host.com*.

Furthermore Request supports asynchronous sending, through the method *BeginGetResponse*. The standard .Net async calling pattern is used.

Code sample 2 demonstrates how to make an async call (without using a callback method) to a mail endpoint.

Code sample 2

```
XmlDocument xdoc = new XmlDocument();
Request raspReq = new Request(
      new Uri("mailto:myMailEndpoint@server.com"));

// Starts the sending
IAsyncResult r = raspReq.BeginGetResponse(
      new RaspMessage(new XmlDocument()), null);

// Ends the sending
Response response = raspReq.EndGetResponse(r);
```

Code sample 1 and Code sample 2 both 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 3.

Code sample 3

```
// 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,
               RaspCredentials credentials,
               SendPolicy sendPolicy);
```

The first constructor in Code sample 3 takes the name of an endpoint in the application configuration file. See the file App.Config in the test project Test_RaspRequest for further reference to how the configured endpoint (OiosiHttpEndpoint) describes the service which we will call.

A configured endpoint is needed no matter what constructor is used, and unless the first constructor in Code sample 3 is used **OiosiHttpEndpoint/OiosiEmailEndpoint is used as a default endpoint, and therefore should always be present in the config file** when using the RASP Library.

The second constructor in Code sample 3 takes an endpoint URI, just like the ones used in Code sample 1 and Code sample 2, but also takes programmatically set certificates (for sending and/or receiving). These will override any certificates given in App.Config. Furthermore a SendPolicy should be given, describing what SOAP action to use for sending (default value is “*”).

5.2 RequestExtended

The RequestExtended request demonstrates the following in addition to the 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.

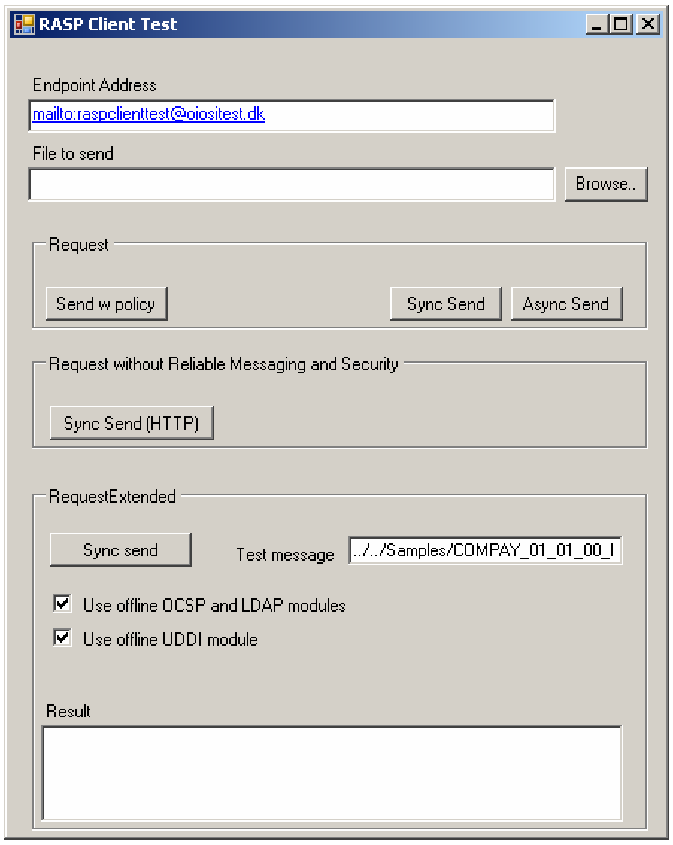
For a concrete example of how to use the Request class, see the test project Test_RaspRequest.

Code sample 4 shows how RequestExtended 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 4

```
// 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();
RaspMessage msg = new RaspMessage(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 RequestExtended test sample by running the Test_RaspRequest project application and pressing “Sync send” in the “RequestExtended” section, see below.



When the application starts, a sample RaspConfiguration.xml file is created. This will be updated dynamically as you check or uncheck the “Use offline OCSP and LDAP module” or “Use offline UDDI module” checkboxes. You can also view the code as an example of generating and dynamically updating configuration files (apart from app.config configuration, which is never updated programmatically).

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 5

```
<ConfigurationSection xsi:type="OcspLookupFactoryConfig">
    <ImplementationNamespaceClass>Oiosi.Rasp.Utilities.Ocsp.OcspLookupTest
    </ImplementationNamespaceClass>
    <ImplementationAssembly>Oiosi.Rasp.RaspLibrary</ImplementationAssembly>
</ConfigurationSection>
<ConfigurationSection xsi:type="LdapLookupFactoryConfig">
      <ImplementationNamespaceClass>
            Oiosi.Rasp.Utilities.Ldap.LdapCertificateLookupTest
      </ImplementationNamespaceClass>
      <ImplementationAssembly>
            Oiosi.Rasp.RaspLibrary
      </ImplementationAssembly>
</ConfigurationSection>
<ConfigurationSection xsi:type="UddiLookupClientFactoryConfig">
      <ImplementationNamespaceClass>
            Oiosi.Rasp.Utilities.Uddi.UddiLookupClientTest
      </ImplementationNamespaceClass>
      <ImplementationAssembly>
            Oiosi.Rasp.RaspLibrary
      </ImplementationAssembly>
</ConfigurationSection>
```

The offline implementations may then be configured differently from the online implementations.

Code sample 6

```
<ConfigurationSection xsi:type="OcspLookupTestConfig">
    <ReturnPositiveResponse>true</ReturnPositiveResponse>
  </ConfigurationSection>
  <ConfigurationSection xsi:type="LdapCertificateLookupTestConfig">
    <StoreLocation>LocalMachine</StoreLocation>
    <StoreName>Root</StoreName>
  </ConfigurationSection>
  <ConfigurationSection xsi:type="UddiLookupClientTestConfig">

<EndpointAddress>http://oiositest.dk/interoptest/oiosiOmniEndpointB.s
vc</EndpointAddress>
    <ExpirationDate>2012-06-01T00:00:00</ExpirationDate>
    <CertificateSubjectSerialNumber>
OID.2.5.4.5=CVR:26769388-UID:1172691221366 + CN=IT- og Telestyrelsen
- OIOSI test virksomhedssignatur, O = IT- og Telestyrelsen //
CVR:26769388, C=DK</CertificateSubjectSerialNumber>
    <TermsOfUseUrl>http://test.dk/termsOfUse.html</TermsOfUseUrl>
    <ServiceContactEmail>test@test.com</ServiceContactEmail>
    <Version>1.0.3</Version>
    <NewerVersionReference />
    <HasNewerVersion>false</HasNewerVersion>
  </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.

5.3 Listener

Oiosi.Rasp.Communication.Listener. is the main class for hosting RASP email services.

Code sample 7

```
// Starts up a service, and adds a callback method for
// receving incoming messages
static void Main(string[] args) {

      Listener listener = new Listener();

      // Add event handling
      listener.MessageReceive +=
            new MessageEventDelegate(MessageReceived);
      listener.ExceptionThrown +=
            new AsyncExceptionThrownHandler(ExceptionThrown);

      listener.Start();

      // Wait until application is terminated . . .

      listener.Stop();
}


public static void MessageReceived(ListenerRequest msg,
      MessageProcessStatus status) {

      // Do something with the incoming message . . .

}
Public static void ExceptionThrown(Exception ex) {
      // Do something with the asynchronous exception
}
```

Code sample 7 shows how to start a standard RASP service up, by reading service information from App.Config. In the configuration file for test project Test_RaspListener one can see how a service, a service behavior and a binding are defined, enabling the code in Code sample 7 to run.

Code sample 7 also demonstrates how a MessageEventDelegate is used to define a callback method to be called whenever a message is received by the listener. The callback method gets a ListenerRequest, which is the incoming message, plus metadata (such as properties set by interceptors), and a status.

Furhtermore an AsyncExceptionThrownHandler is used to catch exceptions raised by the listener. It is strongly recommended to listen to this event, since it might be the one and only chance one has to discover that the service dies.

Code sample 8

```
public Listener (ListenerIdentity listenerIdentity);
```

Listener also offers a constructor, shown in Code sample 8 which takes a ListenerIdentity. Using this constructor allows one to programmatically set certificates that will be used instead of those given in App.config.

ListenerIdentity also allows one to set the transport binding programmatically (to override mail settings for example) and to make listener run a service of another type than the standard RASP implementation.

6 Lesson: Oisi.Rasp.WcfExtensions.Interceptor

There are four interceptors implemented in the current RaspLibrary; the schema validation interceptor, the schematron validation interceptor, the proof of signature validation interceptor and the XSLT transformer interceptor. All are added to the stack in the app.config file. The app.config from earlier in App.Config sample 4 is extended when adding the interceptors.

There is a difference between server side and client side interceptors as they have different functionality. The XSLT transformer interceptor only exists as a server side interceptor. This section describes the server side custom binding and secondly the client side custom binding and finally the extensions.

To insert the server validators in the stack add ServerSchemaValidator, ServerSchematronValidator, ServerXsltTransform and ServerSignatureProof in the binding. The binding with both interceptors can be seen in App.Config sample 5.

The three the ServerSchemaValidator, the ServerSchematronValidator and ServerXsltTransform interceptors have three options:

1. ValidateRequest: If true it will validate the xml on request. Default is true.
2. 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.
3. 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.

The ServerSignatureProof has an option whether it should fault or not if the proof fails. The default is true.
To insert the client validators in the stack add ClientSchemaValidator, ClientSchemaValidator and ClientSignatureProof in the binding. The binding with both interceptors can be seen in App.Config sample 6.

Both the ClientSchemaValidator and the ClientSchematronValidator has two options:

1. ValidateRequest: If true it will validate the xml on request. Default is true.
2. 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.

The ClientSignatureProof has no options.

Finally add the binding elements to bindingElementExtensions as shown in App.Config sample 7.

Note that the ServerSignatureProof and ClientSignatureProof interceptors must be between the RM layer and the security layer.

App.Config sample 5

```
<bindings>
   <customBinding>

        <!-- The OIOSI RASP mail binding -->
        <binding  name="OiosiEmailBinding"
                  closeTimeout="00:01:30"
                  openTimeout="00:01:30"
                  receiveTimeout="00:01:30"
                  sendTimeout="00:01:30">

          <ServerSchematronValidator ValidateRequest="true"
                                     ValidateResponse="true"
                          FaultOnRequestValidationException="true"/>

          <ServerSchemaValidator ValidateRequest="true"
                                 ValidateResponse="true"
                          FaultOnRequestValidationException="true"/>

          <ServerXsltTransformer
                              FaultOnTransformationException="true"
                              PropagateOriginalMessage="true"/>

          <reliableSession/>

          <ServerSignatureProof
            FaultOnRequestValidationException="true"/>

          <security
          messageSecurityVersion="WSSecurity10WSTrustFebruary2005WSSe
          cureConversationFebruary2005WSSecurityPolicy11BasicSecurity
          Profile10"

                    defaultAlgorithmSuite="Default"
                    authenticationMode="MutualCertificate"
                    requireDerivedKeys="false"
                    securityHeaderLayout="Strict"
                    includeTimestamp="true"
                    keyEntropyMode="CombinedEntropy"
                    messageProtectionOrder="SignBeforeEncrypt"
                    requireSignatureConfirmation="false"

           />

           <textMessageEncoding messageVersion="Default"
                 writeEncoding="utf-8"/>

           <OiosiEmailTransport
                outboxImplementation="Oiosi.Rasp.LesnikowskiImplement
                ation.SmtpOutboxLesnikowski,
                Oiosi.Rasp.LesnikowskiImplementation"

                inboxImplementation="Oiosi.Rasp.LesnikowskiImplementa
                tion.Pop3InboxLesnikowski,
                Oiosi.Rasp.LesnikowskiImplementation"

               sendingServerAddress="oiositest.dk"
               receivingServerAddress="oiositest.dk"
               receivingUserName="client"
               receivingPassword="password"
                replyAddress="mailto:client@oiositest.dk"/>
      </binding>
   </customBinding>
</bindings>
```

App.Config sample 6

```
<bindings>
   <customBinding>

        <!-- The OIOSI RASP mail binding -->
        <binding  name="OiosiEmailBinding"
                  closeTimeout="00:01:30"
                  openTimeout="00:01:30"
                  receiveTimeout="00:01:30"
                  sendTimeout="00:01:30">

          <ClientSchemaValidator ValidateRequest="true"
                                 ValidateResponse="true"/>

          <ClientSchematronValidator ValidateRequest="true"
                                     ValidateResponse="true"/>

          <reliableSession/>

          <ClientSignatureProof/>

          <security
          messageSecurityVersion="WSSecurity10WSTrustFebruary2005WSSe
          cureConversationFebruary2005WSSecurityPolicy11BasicSecurity
          Profile10"

                    defaultAlgorithmSuite="Default"
                    authenticationMode="MutualCertificate"
                    requireDerivedKeys="false"
                    securityHeaderLayout="Strict"
                    includeTimestamp="true"
                    keyEntropyMode="CombinedEntropy"
                    messageProtectionOrder="SignBeforeEncrypt"
                    requireSignatureConfirmation="false"

           />

          <textMessageEncoding messageVersion="Default"
                writeEncoding="utf-8"/>

           <OiosiEmailTransport
                outboxImplementation="Oiosi.Rasp.LesnikowskiImplement
                ation.SmtpOutboxLesnikowski,
                Oiosi.Rasp.LesnikowskiImplementation"

                inboxImplementation="Oiosi.Rasp.LesnikowskiImplementa
                tion.Pop3InboxLesnikowski,
                Oiosi.Rasp.LesnikowskiImplementation"

               sendingServerAddress="oiositest.dk"
               receivingServerAddress="oiositest.dk"
               receivingUserName="client"
               receivingPassword="password"
                replyAddress="mailto:client@oiositest.dk"/>
      </binding>
```

App.Config sample 7

```
<extensions>
  <bindingElementExtensions>
    <add name="ClientSchemaValidator"
         type="Oiosi.Rasp.WcfExtensions.Interceptor.Validation.
          Schema.ClientSchemaValidationBindingExtensionElement,
          Oiosi.Rasp.RaspLibrary"/>
    <add name="ServerSchemaValidator"
         type="Oiosi.Rasp.WcfExtensions.Interceptor.Validation.
          Schema.ServerSchemaValidationBindingExtensionElement,
          Oiosi.Rasp.RaspLibrary"/>
    <add name="ClientSchematronValidator"
         type="Oiosi.Rasp.WcfExtensions.Interceptor.Validation.
        Schematron.ClientSchematronValidationBindingExtensionElement,
        Oiosi.Rasp.RaspLibrary"/>
    <add name="ServerSchematronValidator"
         type="Oiosi.Rasp.WcfExtensions.Interceptor.Validation.
        Schematron.ServerSchematronValidationBindingExtensionElement,
        Oiosi.Rasp.RaspLibrary"/>
    <add name="OiosiEmailTransport"
         type="Oiosi.Rasp.WcfExtensions.EmailTransport.
         RaspEmailBindingExtensionElement, Oiosi.Rasp.RaspLibrary"/>
    <add name="ClientSignatureProof"
         type="Oiosi.Rasp.WcfExtensions.Interceptor.
      Security.ClientSignatureValidationProofBindingExtensionElement,
         Oiosi.Rasp.RaspLibrary"/>
    <add name="ServerSignatureProof "
         type="Oiosi.Rasp.WcfExtensions.Interceptor.
      Security.ServerSignatureValidationProofBindingExtensionElement,
         Oiosi.Rasp.RaspLibrary"/>
    <add name=" ServerXsltTransformer"
         type="Oiosi.Rasp.WcfExtensions.Interceptor.XsltTransform.
         ServerXsltTransformationBindingExtensionElement,
         Oiosi.Rasp.RaspLibrary"/>
   </bindingElementExtensions>
 </extensions>
```

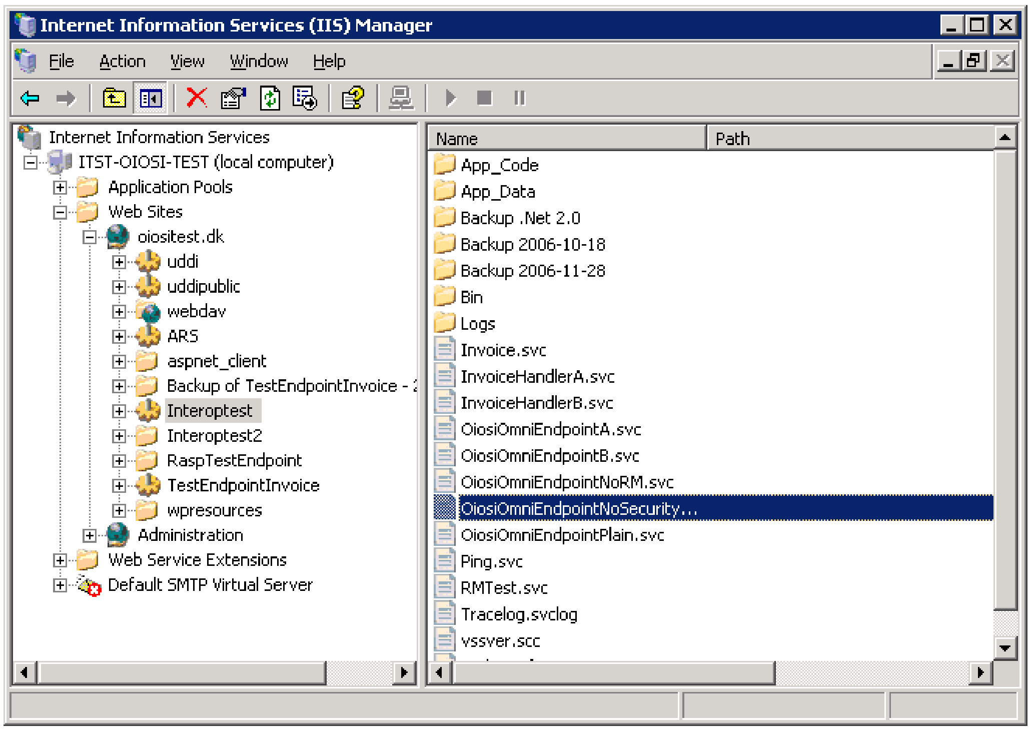
7 Lesson: Setting up an IIS hosted RASP service

The Oiosi.Rasp.Communication.Listener class only supports hosting mail endpoints, but one might want to host an HTTP endpoint, in which case it is recommended that this endpoint should be hosted by Microsoft Internet Information Services (IIS).

A zip file called RaspHTTPEndpointExample.zip is distributed with the RASP Library and can be found under the Resources folder in the RaspLibrary Visual Studio project. In this zip archive 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).

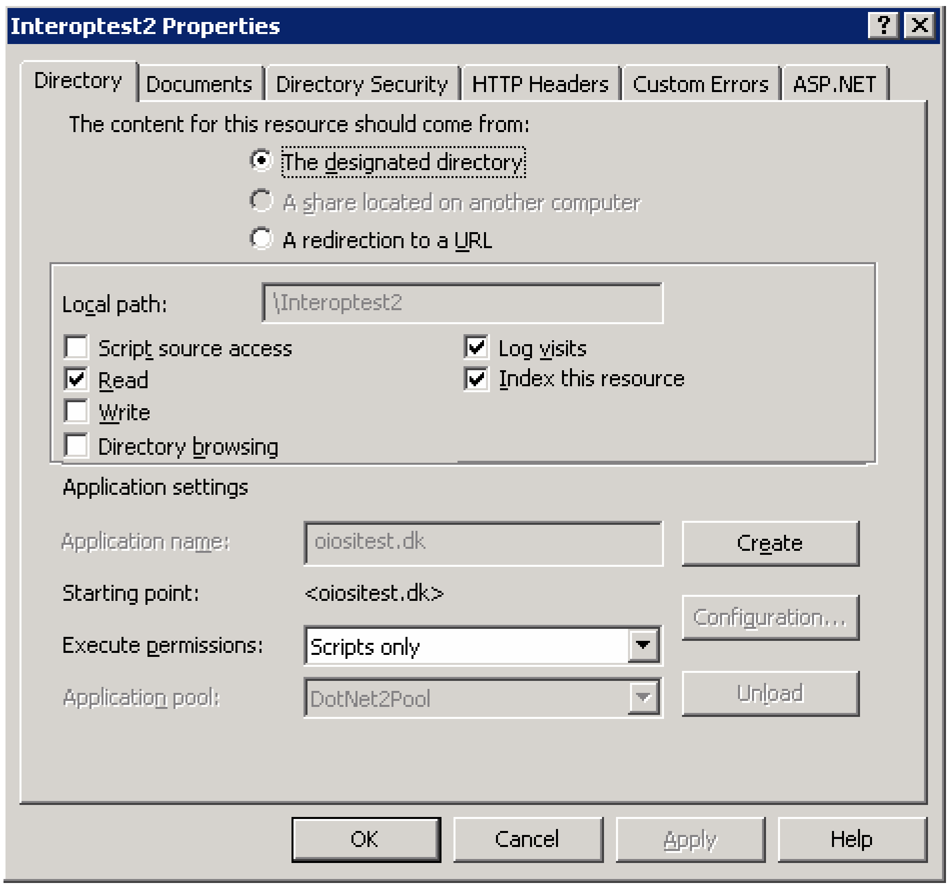
To get started, simply unzip the files into *C:\Inetpub\wwwroot\* (assuming that you have your Windows installation on the C drive).

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



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 unzipped 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

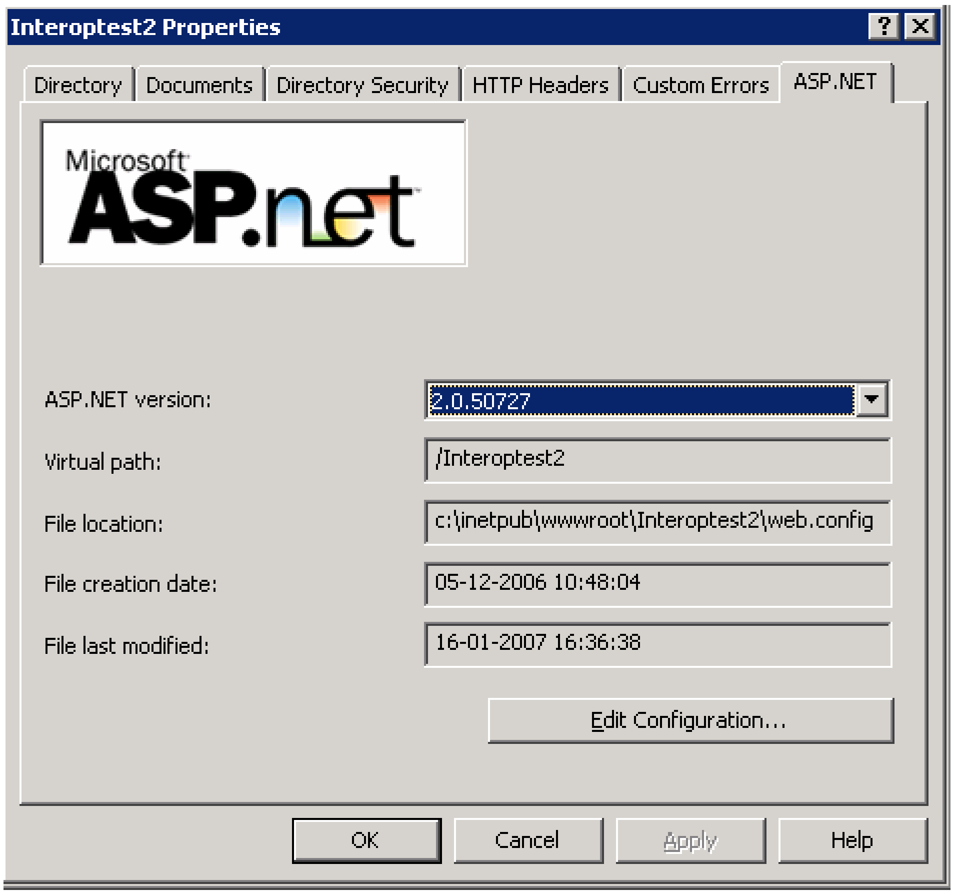
Application name: oiositest.dk

Starting point: <oiositest.dk>

Execute permissions: Scripts only

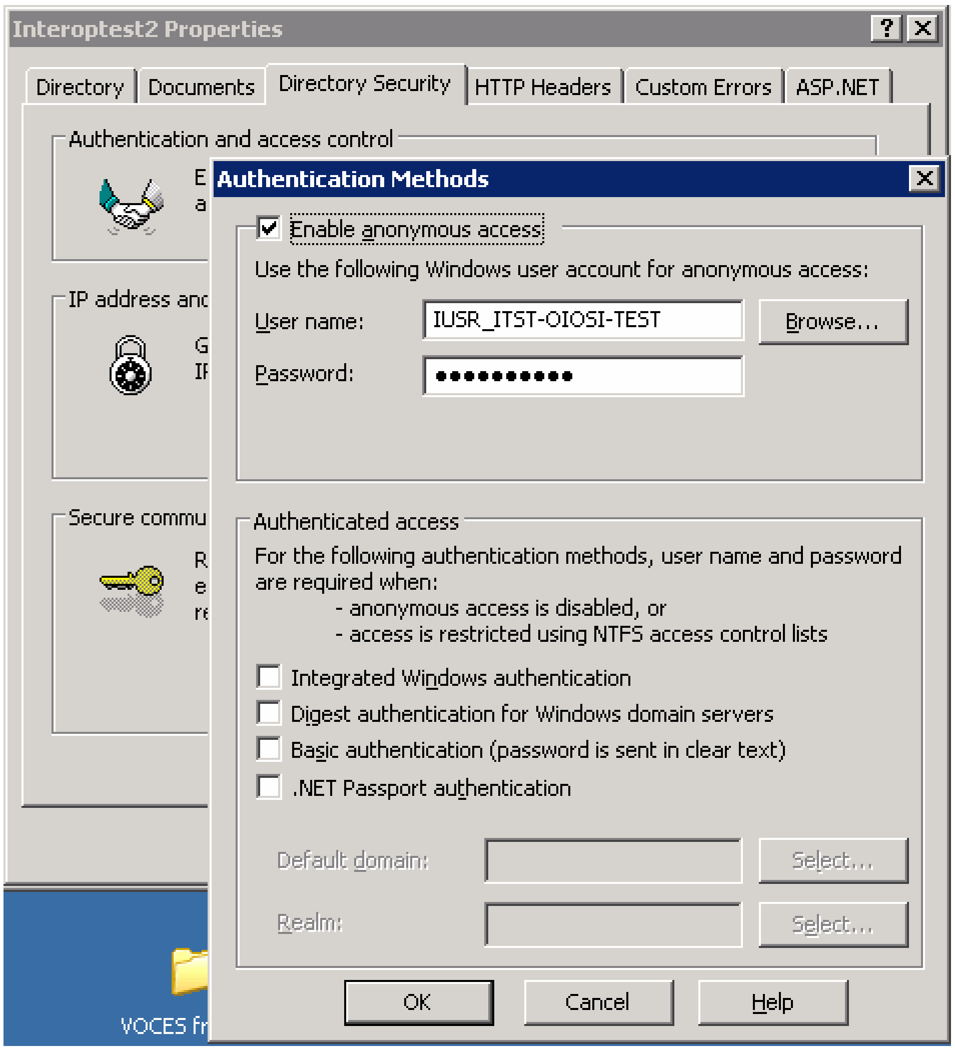
Application pool: DotNet2Pool

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


Interoptest2 Properties

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

IP address and

Secure commu

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) .

7.1.1 Tips for solving common issues when hosting in IIS

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)

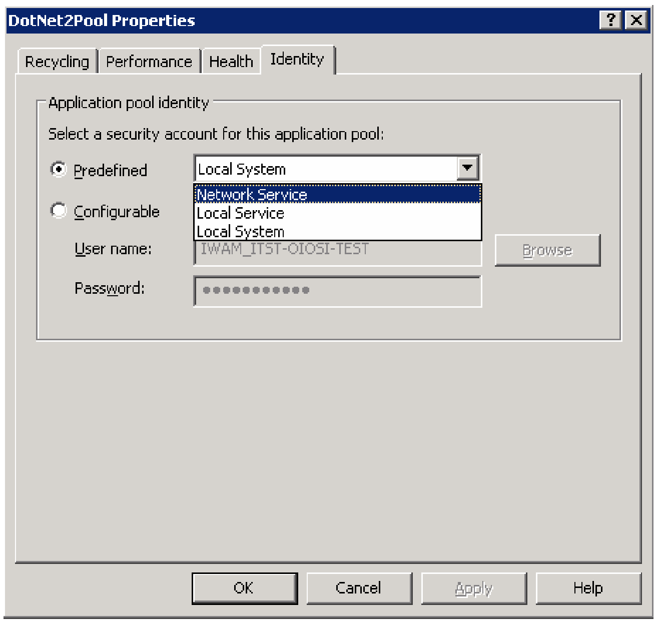
- If that doesn’t work, perhaps the application pool doesn’t have rights to use the certificate. Try:
    - Delete the certificate OiosiTestVOCES.pfx from the MMC window.
    - Run the command line tool WinHttpCertCfg (http://www.microsoft.com/downloads/details.aspx?familyid=c42e27ac-3409-40e9-8667-c748e422833f&displaylang=en)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

      ```
      >WinHttpCertCfg.exe –i OiosiTestVOCES.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 “IT- og Telestyrelsen - OIOSI test virksomhedssignatur” –g –a “aspnet”
      ```

      (where “*IT- og Telestyrelsen - OIOSI test virksomhedssignatur*” is the subject string of the certificate you just imported)
    - In web.config, change the location of the certificate from “Root” to “My” store.



- 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