Tutorial:
Connection of an SQL data source

version: 1.0

Author: David Haasler

Table of contents

Requirements

The developer needs access to the “web.config” of the onion.net Render Engine for this tutorial as well as access to an SQL server.

Description

This tutorial describes the connection of a database - as an SQL data source - to the onion.net Render Engine.

Signs and symbols

Boxes marked with an arrow symbol and a green border contains instruction of what to do next.

This kind of boxes contains tips and tricks.

```
Source code is shown in blue boxes.
```

1 Creating the database

As a first step we will create a database with test data, which we will later configure as a data source in our onion.net Render Engine.

The following script creates a test database, produces a table with test data and generates two saved procedures.

```
CREATE DATABASE SQLTest
GO
USE SQLTest
GO
CREATE TABLE [Contacts](
[id] [int] IDENTITY(1,1) NOT NULL,
[firstname] [nvarchar](50) NOT NULL,
[lastname] [nvarchar](50) NOT NULL,
[age] [int] NOT NULL,
[male] [bit] NOT NULL
) ON [PRIMARY]
GO
INSERT INTO [Contacts] (firstname, lastname, age, male) VALUES ('Walter',
'Webb', 20, 'True')
INSERT INTO [Contacts] (firstname, lastname, age, male) VALUES
('Kristina', 'Kreativ', 25, 'False')
INSERT INTO [Contacts] (firstname, lastname, age, male) VALUES ('Max',
'Mustermann', 30, 'True')
INSERT INTO [Contacts] (firstname, lastname, age, male) VALUES ('Erika',
'Mustermann', 35, 'False')
GO
CREATE PROCEDURE [getContacts] AS
BEGIN
SELECT id, firstname, lastname FROM Contacts
END
GO
CREATE PROCEDURE [getContact] (@id int) AS
BEGIN
DECLARE @count int
SELECT @count = COUNT(1) FROM Contacts WHERE id = @id
IF @count &gt; 0
SELECT id, firstname, lastname, age, male FROM Contacts WHERE id = @id
```

```
ELSE
RAISERROR ('no data found', 16, 1)
END
GO
```

2 Configuring the onion.net Render Engine

After the test data has been created in the SQL server, we can now configure this database as a data source in the “web.config” of our onion.net Render Engine. The following configuration section is inserted within the element “dataSources”.

```
<source xlinkPrefix="sql"
type="Onion.RenderEngine.CommonDataSources.MsSqlServerRepository,
Onion.RenderEngine.CommonDataSources">
  <server connectionString="user
id=username;password=password;server=localhost;initial catalog=SQLTest"
typeIdentifier="SQL">
    <commands>
      <command name="getContacts" contentType="ContactList">
        <dataViews>
          <list
                    query="exec getContacts"
                    fields="id firstname lastname"
                    resultType="sql"
                    dependencies="Contacts"
          />
        </dataViews>
      </command>
      <command name="getContact" contentType="Contact">
        <dataViews>
          <detail
                    query="exec getContact @id"
                    fields="id firstname lastname age male"
                    resultType="sql"
                    dependencies="Contacts"
          />
        </dataViews>
        <queryParameters>
          <parameter name="id" type="int" />
        </queryParameters>
      </command>
    </commands>
  </server>
</source>
```

As you may see in the configuration section, a further data source is defined. “sql” has now been configured as a prefix.

The connection to the SQL server has been configured within the sources-element. In this case, the SQL server runs on the local system.

Two commands are now created for this server. The first command “getContacts” returns the results as the data type “ContactList”. The saved method “getContacts” is called here, which is to return the columns “id”, “firstname” and “lastname”.

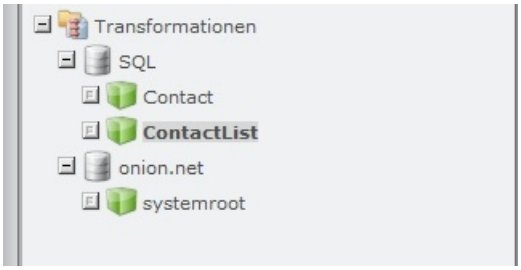
The second command “getContact” is to only return a contact; the return type is therefore “Contact”. As a saved procedure, “getContact” is called, which is transferred the parameter “@id”. This parameter is mapped under the section “queryParameter” to a data type (the “@id” to “int” in the example). As well as “id”, “firstname” and “lastname”, the saved procedure is to also return “age” and “male”.

3 Creating XSL templates

Now we would like to represent the test data in the database by means of the onion.net Render Engine.

The goal is to create an overview and a detail page for the contacts in the database.

At the end of the tutorial the structure tree should look as in the illustration.



For this purpose, the following template is created for the root-document:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sql="http://onionworks.net/2006/datasources/mssql"
  xmlns:onion="http://onionworks.net/2004/data"
  version="1.0"
>
  <xsl:param
    name="id"
    c.optional="true"
    c.type="Number"
  />
  <xsl:output
    method="xml"
    omit-xml-declaration="yes"
    indent="no"
  />
  <xsl:template match="/">
    <xsl:choose>
      <xsl:when test="$id">
       <c.literalCall id="{sql:createCommandXLink('sql', 'getContact', $id)}"
method="detail" />
      </xsl:when>
      <xsl:otherwise>
        <c.literalCall
                id="{sql:createCommandXLink('sql', 'getContacts')}"
                method="list"
                delegate="{c.xlink('default')}"
        />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

default Template

The following templates are no longer created under the data source onion.net, but under the data source SQL. ContentTypes, which are created in the data source as data types, have been defined in the “web.config”. This would be the data types ContactList and Contact.

For the overview page, the method “list” is called for the data type ContactList. The template lists the results of the query.

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:onion="http://onionworks.net/2004/data"
  version="1.0"
>
  <xsl:param name="delegate" />
  <xsl:output
    method="xml"
    omit-xml-declaration="yes"
    indent="no"
  />
  <xsl:template match="/">
    <table>
      <tr>
        <th>ID</th>
        <th>Firstname</th>
        <th>Lastname</th>
      </tr>
      <xsl:for-each select="result/row">
        <tr>
          <td>
            <xsl:value-of select="@id" />
          </td>
          <td>
            <a href="{c.literalUri($delegate, 'id', @id)}">
              <xsl:value-of select="@firstname" />
            </a>
          </td>
          <td>
            <a href="{c.literalUri($delegate, 'id', @id)}">
              <xsl:value-of select="@lastname" />
            </a>
          </td>
        </tr>
      </xsl:for-each>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

Finally, we create the template for the detail view. The method “detail” is now called for the data type “Contact”.

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:onion="http://onionworks.net/2004/data"
  version="1.0"
>
  <xsl:output
    method="xml"
    omit-xml-declaration="yes"
    indent="no"
  />
  <xsl:template match="/result">
    <xsl:for-each select="row">
      <h1>
        <xsl:value-of select="concat(@firstname, ' ', @lastname)" />
      </h1>
      <ul>
        <li>Age:
          <xsl:value-of select="@age" />
        </li>
        <li>Male:
          <xsl:value-of select="@male" />
        </li>
      </ul>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```