Installation und Benutzung der Templates: ASP.NET Web Part Testumgebung zur Web Part-Entwicklung

Einführung

Web Parts zu entwickeln ist erst seit der Einführung von Microsoft Office Sharepoint Server 2007 in vielen Firmen so richtig ein Thema geworden. Allerdings ist die Entwicklung von Sharepoint Web Parts auf dem „offiziellen“ Weg nicht ohne Tücken. Man braucht eigentlich einen Windows Server mit mindestens den „Windows Sharepoint Services 3.0“ oder ein entsprechendes Image in einer virtuellen Umgebung wie Virtual PC oder VMWare. So ein Image braucht dann aber so an die 2GB Speicher, damit es vernünftig läuft. Also muss die Maschine auf der so ein Image dann betrieben wird mindestens 4GB Arbeitsspeicher haben. Dann kann man die „Visual Studio Extensions for Windows Sharepoint Services“ kurz VSEWSS dort installieren. Dann erst hat man in Visual Studio die entsprechenden Project Templates zur Verfügung. Wie man die VSEWSS auch auf Vista oder XP auf „inoffiziellem“ Weg installieren kann, habe ich in meinem Blog beschrieben: http://blogs.msdn.com/martinv.

Allerdings gibt es Web Parts auch in ASP.NET seit der Version 2.0 und da die Sharepoint Services auf ASP.NET basieren sind ASP.NET Web Parts (mit einigen kleinen Einschränkungen) auch auf einer Sharepoint Umgebung installierbar und natürlich auch benutzbar. Also lassen sich Web Parts auch in einer Umgebung ohne Sharepoint Services entwickeln, sofern eben keine Objekte oder APIs aus dem Sharepoint Bereich benutzt werden. Also lassen sich in dieser Testumgebung Web Parts für ASP.NET und Sharepoint Server entwickeln, die z.B. Daten aus anderen Systemen oder Datenbanken visualisieren, oder z.B. Web Services benutzen um so Informationen anzuzeigen usw.

Technische Voraussetzungen

- Windows Vista, Windows XP SP2, Windows Server 2003 oder Windows Server 2008
- Mindestens .NET Framework 3.0
- Visual Studio 2005, Visual Studio 2008 (Express, Standard, …)
- SQL Server oder SQL Server Express

Installation

Die Installation ist denkbar einfach. Man muss nur die beiden Dateien „WebPartTestEnvInstaller.vsi“ und „WebPartTemplateInstaller.vsi“ herunterladen. Es handelt sich hier bei um Visual Studio Templates, die sich einfach durch einen Doppelklick auf die Dateien installieren lassen.
Die Datei „WebPartTestEnvInstaller.vsi“ enthält das Project Template für die ASP.NET Web Part Testumgebung, sowie ein Projekt mit Demo Web Parts. Mit Hilfe dieser Demo Web Parts lassen sich die Funktionalitäten der Testumgebung ausprobieren. Dazu dann aber später mehr …
Die Datei „WebPartTemplateInstaller.vsi“ enthält Project Template für ASP.NET Web Parts. Damit lassen sich dann schnell eigene ASP.NET Web Parts erstellen.

Die ASP.NET Web Part Testumgebung

1. Voraussetzungen:

Damit die Testumgebung funktioniert müssen die ASP.NET Application Services eingerichtet sein. Die Testumgebung benötigt die Personalization Services. Um diese Services zu nutzen benötigt man einen entsprechenden Provider, mitgeliefert wird eine Implementierung für den Microsoft SQL Server (Express). Dieser benötigt dann eine bestimmte Datenbank, welche man auf unterschiedliche Weise nutzen kann.

Mit SQL Server Express Installation:

Hier hat man zwei Möglichkeiten, eine zentrale Datenbank (normalerweise: aspnetdb) im SQL Server Express einrichten, die dann von allen ASP.NET Applikationen genutzt werden kann. Oder man kann eine lokale aspnetdb.mdf Datei an den SQL Express „attachen“. Die Datei wird dann in App_Data Verzeichnis angelegt.

a. Lokale aspnetd.mdf Datei im App_Data Verzeichnis:

Dies ist eigentlich voreingestellt und ändert sich auch nicht mit Einrichtung der Datenbank im SQL Server. Diese Voreinstellung findet man in der Datei „machine.config“, welche Voreinstellungen für alle ASP.NET Applikationen enthält. Die Datei befindet sich im Ordner: „C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG“. Hier findet man folgenden Eintrag:

```
  <connectionStrings>
    <add name="LocalSqlServer" connectionString="data
source=.\SQLEXPRESS;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User
Instance=true" providerName="System.Data.SqlClient"/>
  </connectionStrings>
```

Wenn in nun in der lokalen web.config die Personalization Services für Web Parts aktiviert sind und bzgl. einer Datenbank mit dem Namen „LocalSqlServer“ kein die

Voreinstellung überschreibender Eintrag gemacht wurde. Wird bei ausführen der APS.NET Applikation automatisch die Datei aspnetdb.mdf, welche die Datenbank aspnetdb enthält, erzeugt. Das „Einschalten der Personalization Servcies für Web Parts sieht in der lokalen web-config so aus:

```
  <webParts enableExport="true">
    <personalization>
      <authorization>
        <allow users="domain\username" roles="admin"
          verbs="enterSharedScope" />
      </authorization>
    </personalization>
  </webParts>
```

Natürlich kann ich diese Einstellung auch in meiner lokalen web.config vornehmen, so dass in jedem Fall eine lokale *.mdf Datei im App_Data Verzeichnis verwendet werden soll, selbst wenn die Voreinstellung in der web.config verändert wurde. Dies würde man dann so machen:

```
  <connectionStrings>
    <remove name”LocalSqlServer”="" />
    <add name="LocalSqlServer" connectionString="data
source=.\SQLEXPRESS;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User
Instance=true" providerName="System.Data.SqlClient"/>
  </connectionStrings>
```

b. Zentrale Datenbank im SQLServer (Express) :

Um diese Services einzurichten kann man das Werkzeug aspnet_regsql.exe auf zwei Arten verwenden. Die Applikation befindet sich im Ordner: „C:\Windows\Microsoft.NET\Framework\v2.0.50727“.

Als Windows Applikation im Wizard Modus. Dazu die Datei Doppelklicken oder von der Kommando Zeile mit „aspnet_regsql –W“ starten. Bei der ersten Seite steht nur was das Tool macht, also mit Next auf die nächste Seite gehen und dort sicherstellen dass „Configure SQL Server for application services“ ausgewählt ist und weiter mit Next. Hier jetzt den Namen des SQL Servers angeben den man verwenden möchte. Im Fall einer lokalen SQLEXPRESS Installation sieht das Ganze dann so aus:

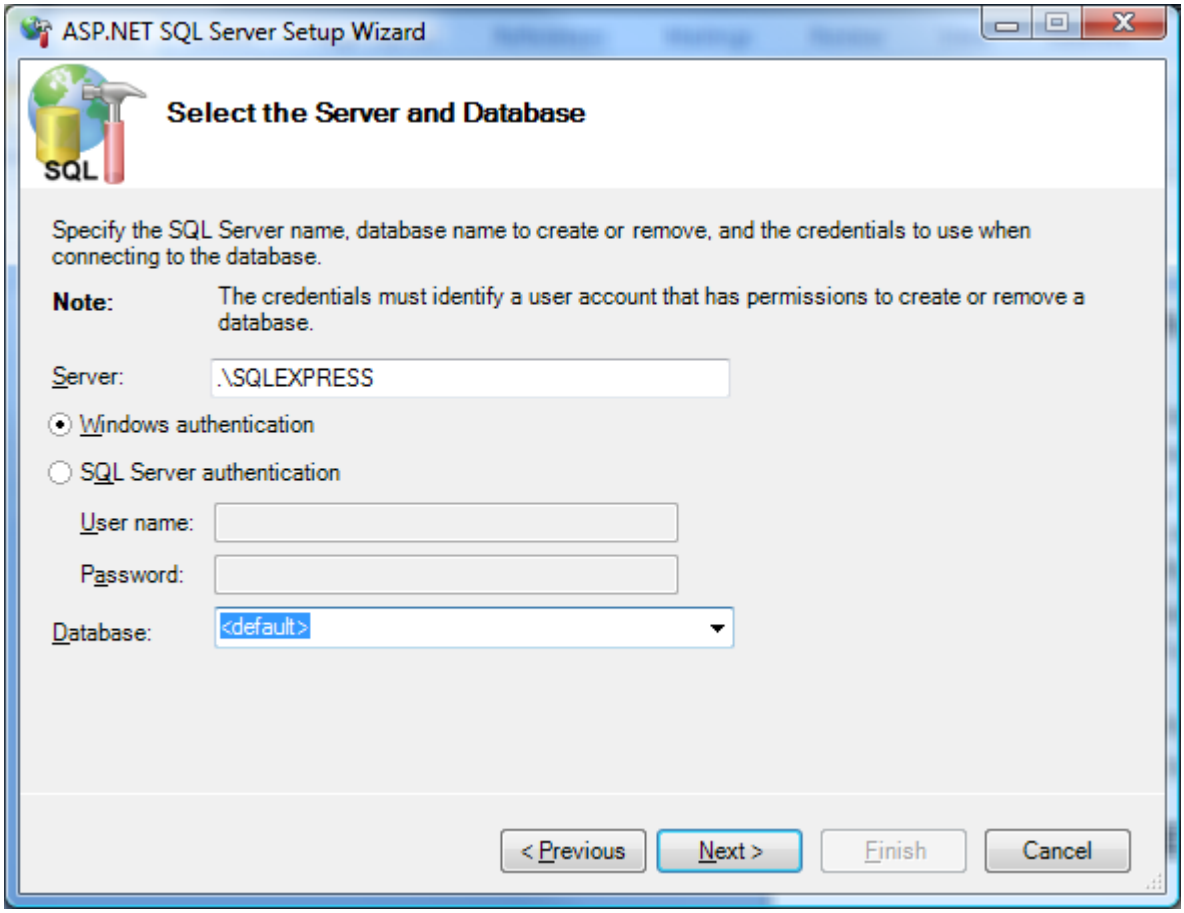

Abbildung 1: SQL Server Konfiguration

Von der Kommandozeile. Hierzu ein Kommandozeilenfenster öffnen und folgendes Kommando eingeben, wenn man eine lokale Installation von SQLEXPRESS hat:

```
C:\> aspnet_regsql -E -S .\SQLEXPRESS -A all
```

Mit einer vollwertigen SQL Server Installation:

Normalerweise installiert man die Datenbank „aspnetdb“ im SQL Server. Da eine lokale aspnetdb.mdf Datei hier wenig Sinn macht, da man dann nur eine Datei attachen kann, weil der SQL Server das Attribut „`User Instance=true`" nicht unterstützt. Somit kann man nur eine Datenbank „aspnetdb“ betreiben. Also sollte hier immer mit „aspnet_regsql.exe“ eine Datenbank für die Services angelegt werden. Das Vorgehen ist dann dasselbe wie bei der zentralen Variante bei SQL Express. Bei einer Default Installation des SAL Servers, gibt man nur den Servernamen an oder man kann auch alternativ dazu „(local)“ eintragen. Im Falle eine named instance des SQL Servers muss diese entsprechend hier eingetragen werden z.B.: <Maschinenname\Instanzname>.

Im Falle einer der Verwendung einer named instance des SQL Server sieht dann der Kommandozeilenaufruf etwa so aus:

```
C:\> aspnet_regsql -E -S Servername\Instanzname -A all
```

Für den Fall einer SQL Server 2005 oder 2008 Konfiguration mit Default Installation:

```
C:\> aspnet_regsql -E -S (Servername) -A all
```

Verwendung der zentralen Datenbank:

Wenn man die zentrale aspnetdb Datenbank im SQL Server (Express) verwenden möchte kann man die Voreinstellung in der machine.config ändern oder natürlich auch in der lokalen web.config einstellen:

```
  <connectionStrings>
    <remove name="LocalSqlServer" />
    <add name="LocalSqlServer"
connectionString="Server=.\SQLEXPRESS;Database=aspnetdb;Integrated
Security=true" providerName="System.Data.SqlClient" />
  </connectionStrings>
```

Bei einer Installation eines „vollwertigen“ SQL Servers ist dann beim Server-Eintrag im Connectionstring der Servername, (local) oder einfach „.“ Einzutragen bei einer named instance ist dann das Muster Servername\Instanzname:

```
  <connectionStrings>
    <remove name="LocalSqlServer" />
    <add name="LocalSqlServer"
connectionString="Server=Server\Instanz;Database=aspnetdb;Integrated
Security=true" providerName="System.Data.SqlClient" />
  </connectionStrings>
```

2. Anlegen der Solution

In Visual Studio auf Project->New klicken. Dann Visual C# und Web als Projekttyp auswählen. Unter „My Templates“ das Template „APS.NET Web Part Test Environment“ selektieren. Name für das Projekt eintippen und dann auf Ok klicken.

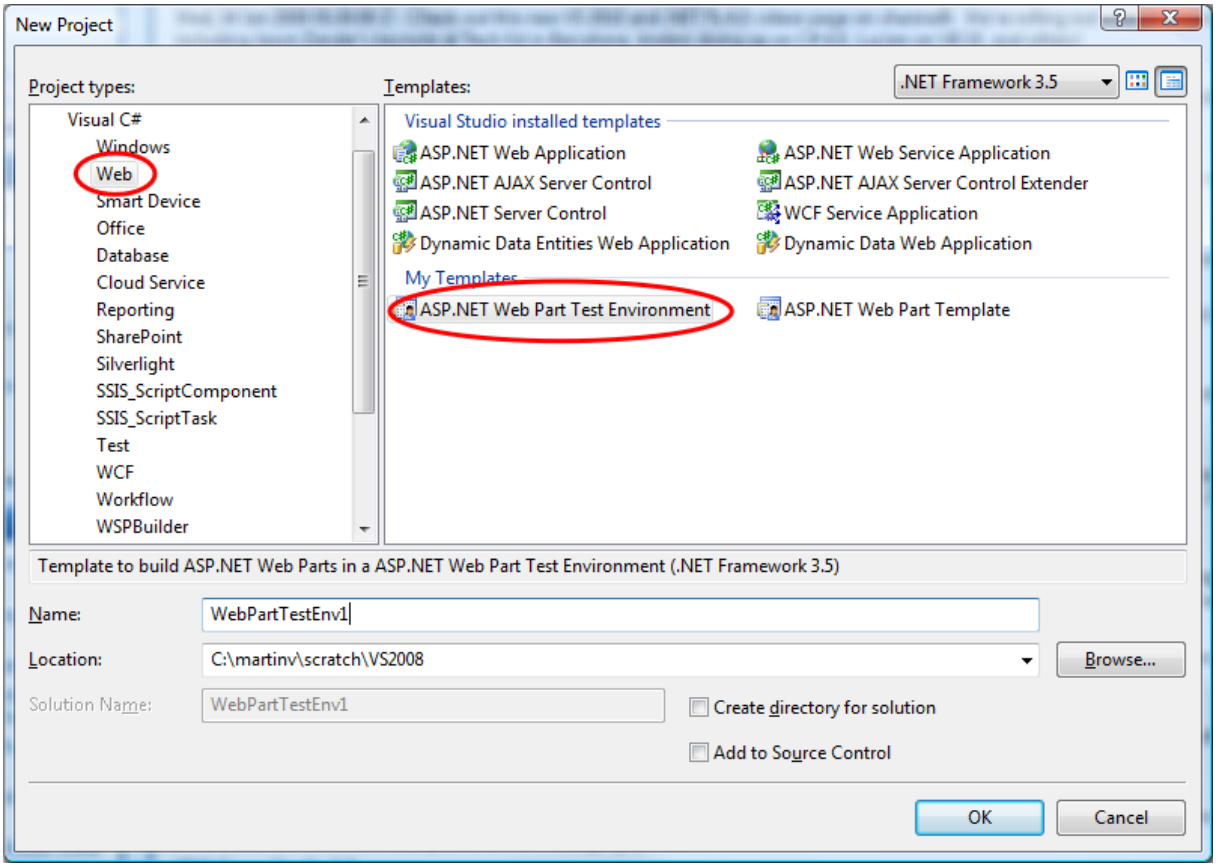

Abbildung 2: Projekt Dialog mit eigenen Templates

Nachdem die Solution mit den zwei Projekten angelegt wurde, muss noch der richtige Connection String in der web.config Datei eingetragen werden damit die ASP.NET „Personalization Services“ funktionieren. Je nachdem ob SQL Express oder ein vollwertiger SQL Server verwendet wird, die entsprechende Zeile auskommentieren. Oder im Falle einer Installation des SQL Servers in einer „named instance“ entsprechend anpassen.

```
<connectionStrings>
    <remove name="LocalSqlServer" />
    <!--<add
connectionString="Server=.\SQLEXPRESS;Database=aspnetdb;Integrated
Security=true" name="LocalSqlServer"
providerName="System.Data.SqlClient" />-->
    <add connectionString="Server=(local);Database=aspnetdb;Integrated
Security=true" name="LocalSqlServer"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

Um zu überprüfen ob die „Personalization Services“ auch funktionieren sollte man noch das APS.NET Konfigurationstool aufrufen. Dazu findet man wenn im Solution Explorer, wenn das Web Projekt selektiert ist einen Button in der Toolbar:

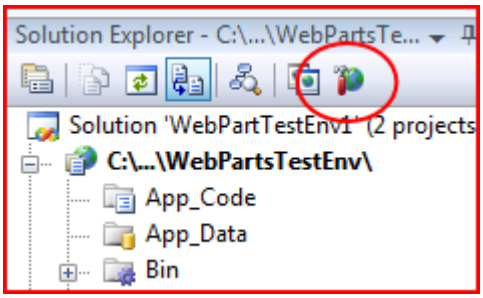

Abbildung 3: ASP.NET Konfiguration starten

Nachfolgend wird eine Web Page gestartet, welche es erlaubt die ASP.NET Applikation interaktiv zu konfigurieren. Uns interessiert aber nur die Provider- oder Anbieterkonfiguration.

Abbildung 4: ASP.NET Websiteverwaltung I

Nach dem klicken auf einen der beiden markierten Links, erscheint dann diese Seite:

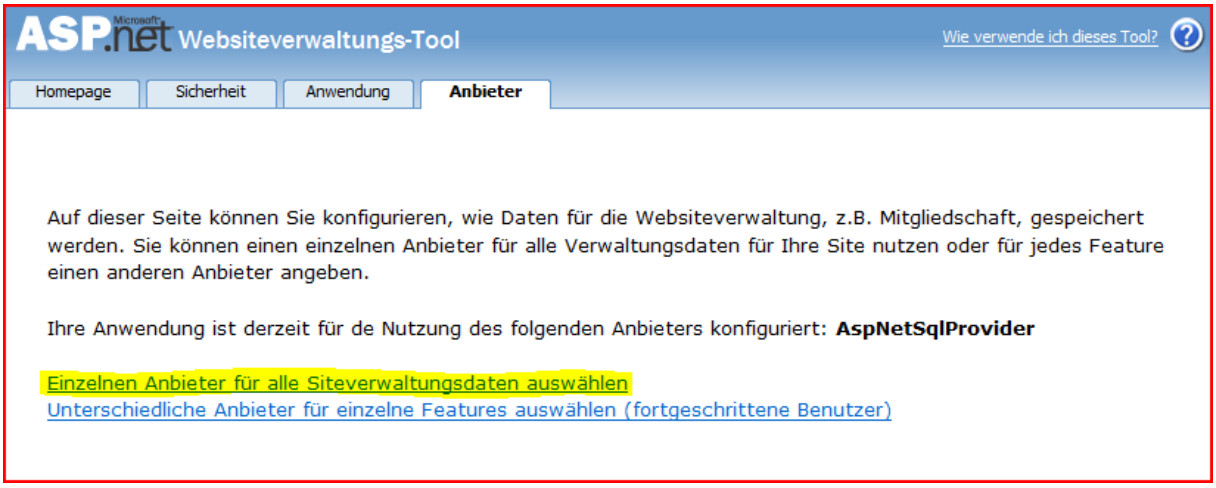

Abbildung 5: ASP.NET Websiteverwaltung II

Weiter geht es mit einem Klick auf den markierten Link und man kommt auf eine Seite, die es ermöglicht den Kontakt zur Datenbank für die „Personalization Services“ (aspnetdb) zu testen.

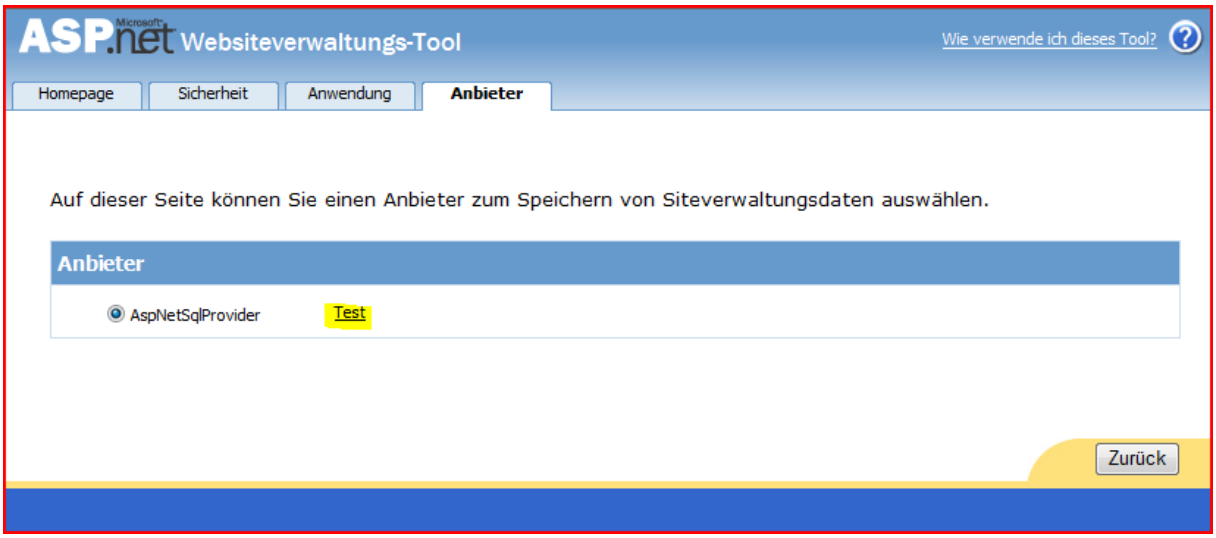

Abbildung 6: ASP.NET Websiteverwaltung III

Nach dem Klick auf den Test Link sollte dann diese Meldung erscheinen:

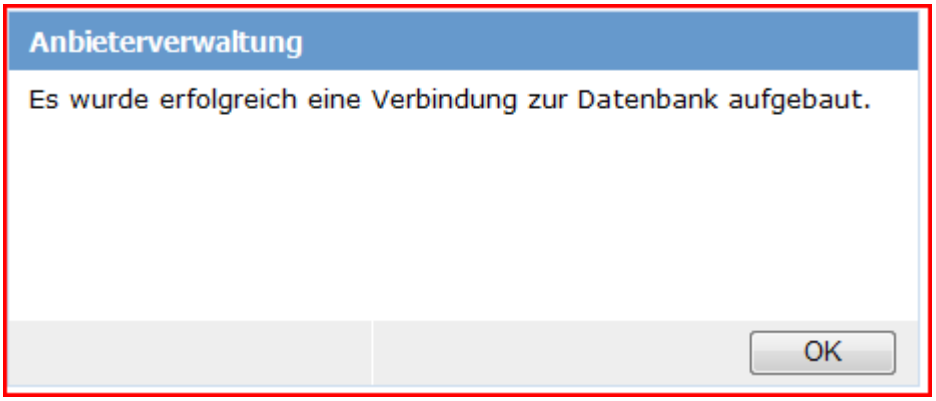

Abbildung 7: ASP.NET Websiteverwaltung - Datenbank Test erfolgreich

Sollte diese Meldung kommen:

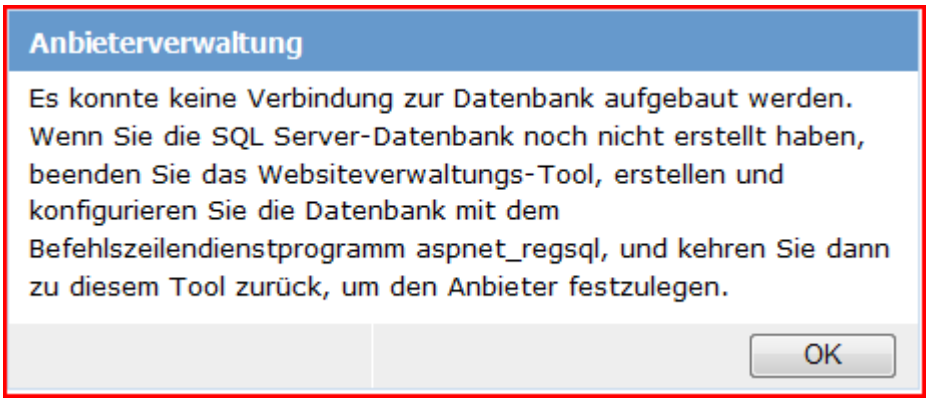

Abbildung 8: ASP.NET Websiteverwaltung - Datenbank Test fehlerhaft

In diesem Fall sollten die Einstellungen bzgl. der Datenbank in der web.config überprüft werden.

3. Evaluierung der Testumgebung

Nach dem die Verbindung zur Datenbank erfolgreich überprüft ist kommen wir jetzt zu eigentlichen ASP.NET Applikation. Um zu zeigen welche Einstellungsmöglichkeiten in einer Web Part Testumgebung getestet werden könne sind in dem Template auch bereits vier einfache Web Parts mitgeliefert. Zwei dieser Web Parts sind auch in der Testseite eingebunden. Wenn man die Anwendung <Steuerung> +F5 startet oder über das Menu „Debug-> Start Without Debugging“ in Visual Studio:

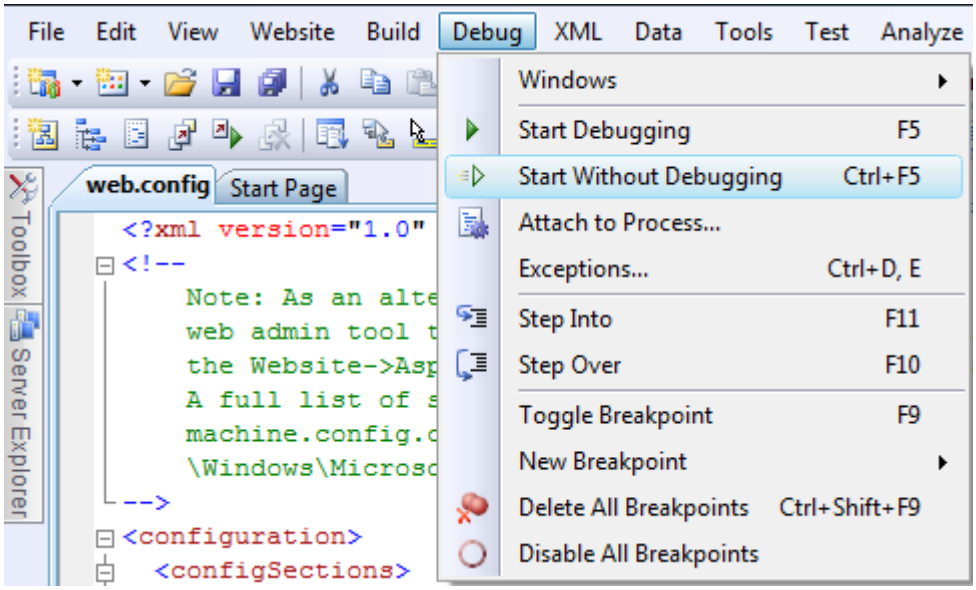

Abbildung 9: Starten der ASP.NET Anwendung ohne Debugger

Nach dem Start der Applikation erscheint dieses Bild im Browser. In dem oberen grau hinterlegten User Control, lässt sich der Display Mode des WebPart-Managers umstellen.

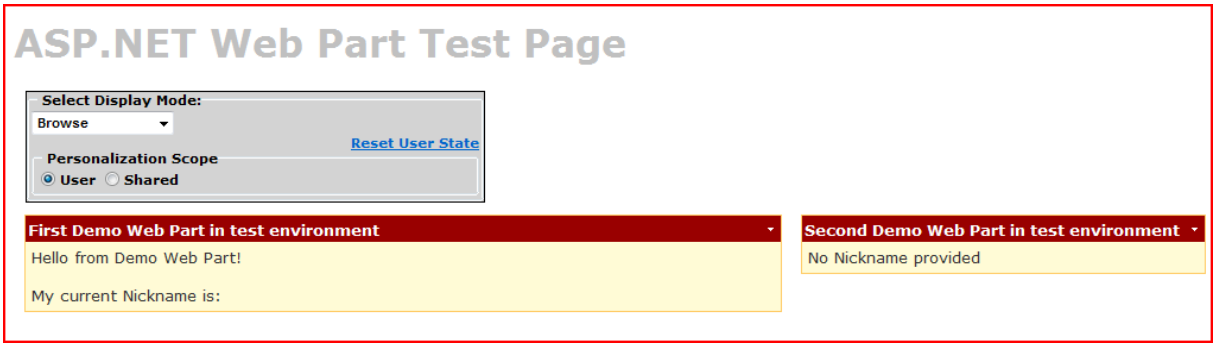

Abbildung 10: Screenshot der ASP.NET Anwendung

Als erstes betrachten wir den Edit Mode, dazu in der DropDown Box Edit auswählen. Darauf ändert sich die Darstellung, so dass die beiden Web Part Zonen mit Umrandung dargestellt werden:

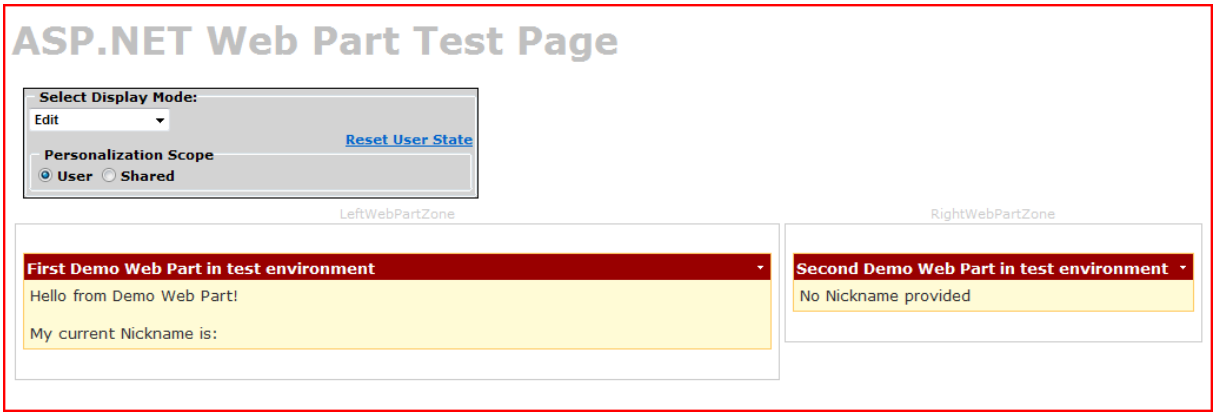

Abbildung 11: Screenshot der ASP.NET Anwendung - Edit Mode I

Jetzt lassen sich die Web Parts beliebig in die beiden Web Part Zonen per „Drag and Drop“ verschieben, dies ist übrigens dasselbe Verhalten wie im „Design Mode“. Im „Edit Mode“ kommt aber noch weitere Funktionalität hinzu. Wenn man im ersten Web Part in der linken Web Part Zone auf den kleinen Pfeil nach unten klickt ist dort ein weiterer Eintrag „Edit“ hinzugekommen.

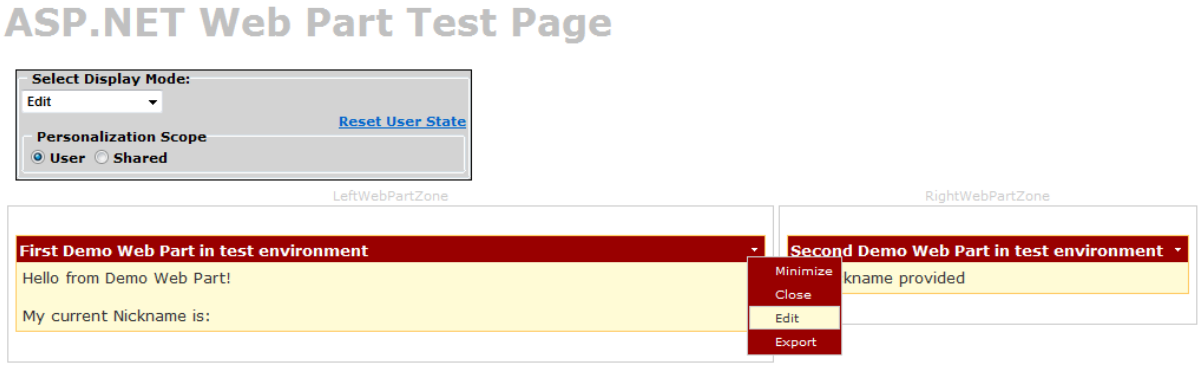

Abbildung 12: Screenshot der ASP.NET Anwendung - Edit Mode II

Wenn man auf diesen „Edit“ Eintrag im Menü klickt erscheint in der Web Seite die Web Part Editor Zone:

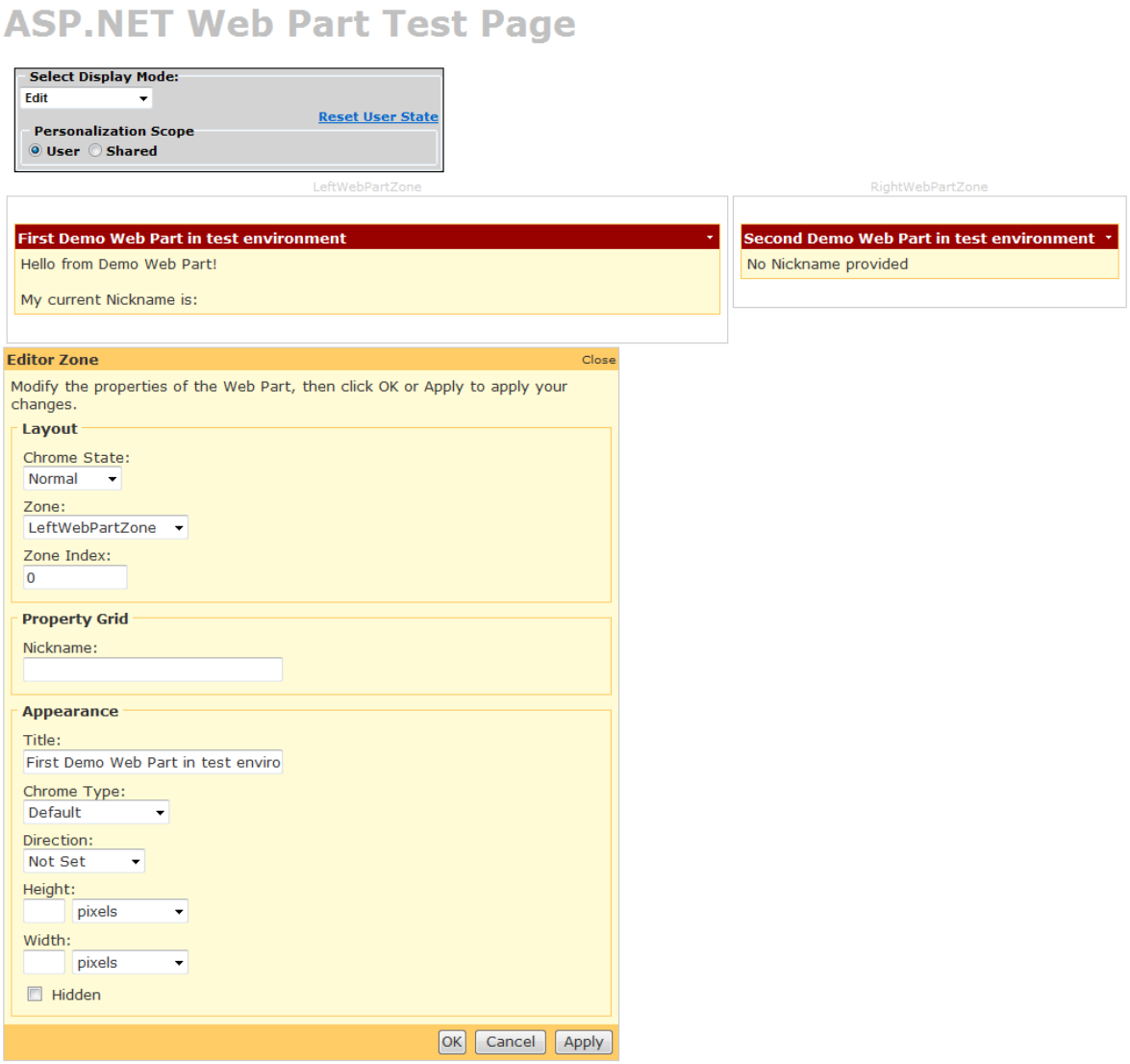

Abbildung 13: Screenshot ASP.NET Anwendung - Edit Mode III

Hier lassen sich jetzt diverse Einstellungen für das Web Part vornehmen. Wobei es sich bei den Einträgen unter Layout und Appearance um Standardeinstellungsmöglichkeiten handelt. Im Bereich werden Property Grid werden Einstellungen dem User zur Verfügung gestellt die der Entwickler als erweiterte Einstellungen implementiert hat dies sieht in diesem Fall im Code so aus:

```
        private string _userNickName = String.Empty;

        // Add the Personalizable and WebBrowsable attributes to the
        // public properties, so that users can save property values
        // and edit them with a PropertyGridEditorPart control.
        [Personalizable(), WebBrowsable, WebDisplayName("Nickname")]
        public String NickName
        {
            get
            {
                object o = ViewState["NickName"];
                if (o != null)
                    return (string)o;
                else
                    return _userNickName;
            }

            set { _userNickName =
System.Web.HttpContext.Current.Server.HtmlEncode(value); }
        }
```

Auf diese Weise lassen sich Properties im Web Part über die ASP.NET Anwendung personalisieren. Wenn man jetzt hier einen neuen „Nickname“ eingibt erscheint dieser auch im Web Part. Also beliebigen String eintippen und in der Editor Zone auf Ok klicken.

Um ein weiteres Feature welches hier gezeigt werden soll zu aktivieren, muss in der DropDown Box der Display Mode auf Connect eingestellt werden. Jetzt kann man eine Connection zwischen zwei Web Parts einrichten, damit diese über ein implementiertes Interface Daten austauschen können. Dies geschieht wie beim Aufruf des Edit Modes. Also im linken Web Part wieder auf den kleinen Pfeil rechts oben klicken und Connect im Menü auswählen. Somit erscheint nun auch die Connections Tone in der Seite.

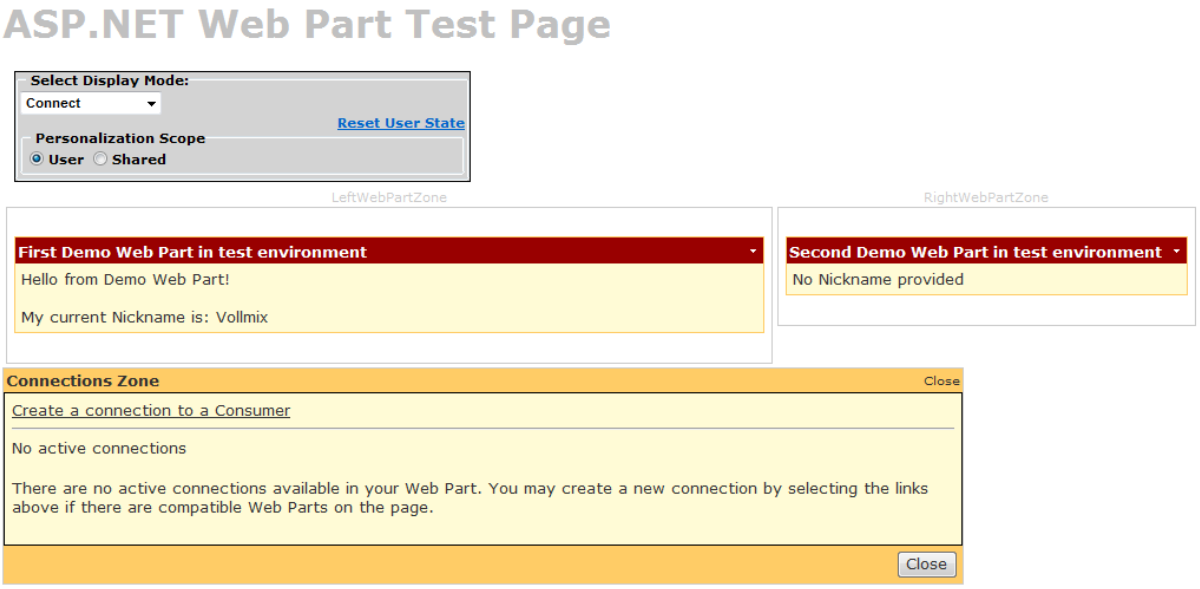

Abbildung 14: Screenshot ASP.NET Anwendung - Connect Mode I

Jetzt einfach auf den Link “Create a connection to a Consumer” klicken.

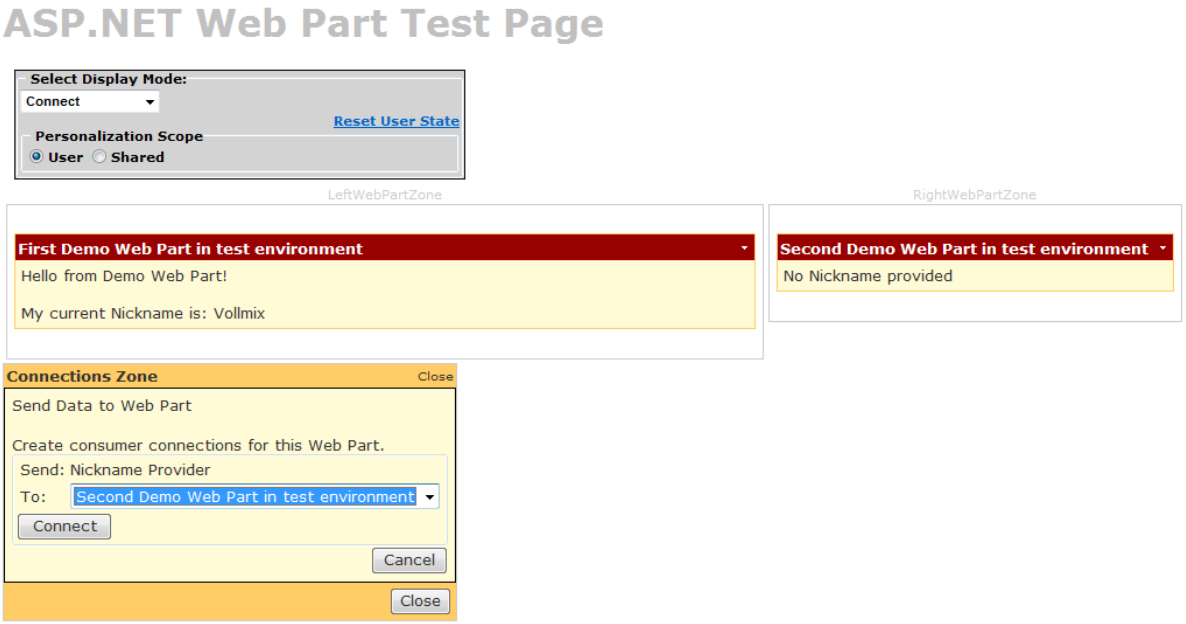

Abbildung 15: Screenshot ASP.NET Anwendung - Connect Mode II

Dann in der Drop Down Box “Second Demo Web Part in test environment” auswählen und auf Connect klicken.

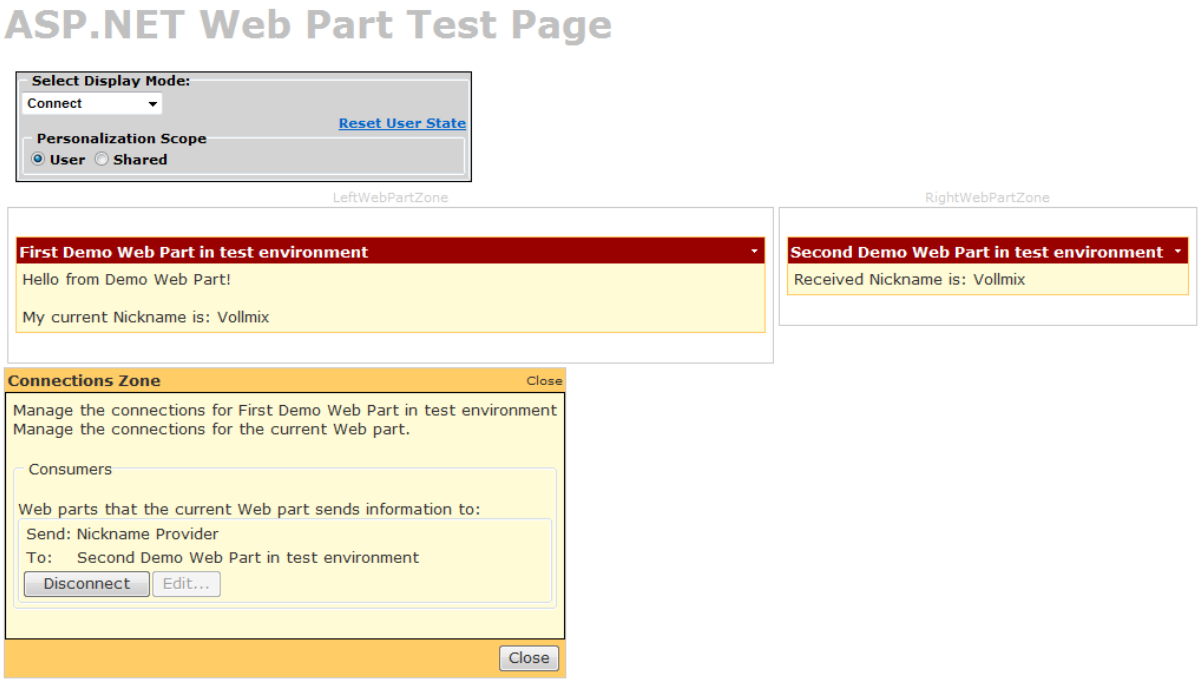

Abbildung 16: Screenshot ASP.NET Anwendung - Connect Mode III

Jetzt erscheint der eingegebene „Nickname“ auch im rechten Web Part, da die beiden jetzt über ein Interface kommunizieren können. Um dies zu erreichen wurde ein Interface zur Kommunikation implementiert:

```
using System;
/// <summary>
/// Interface declaration for the Converstaion between Web Parts
/// </summary>
public interface IMyConnection
{

    String CurrentNickname
    {
        get;
        set;
    }
}
```

Im „Provider“ wird dann das Interface implementiert:

```
public class FirstDemoPart : WebPart, IMyConnection
{
    private String ConversationNickname = String.Empty;



    // Interface for the Connection to the second Web Part
    [ConnectionProvider("Nickname Provider")]
    // Method name doesn't matter
    public IMyConnection ProvideNickName()
    {
        return this;
    }

}
```

Auf der Consumer Seite sieht die Implementierung so aus:

```
public class SecondDemoPart : WebPart
{
    private IMyConnection _myconn;


    [ConnectionConsumer("Nickname Provider")]
  // Method name doesn't matter
    public void GetConversationConsumer(IMyConnection info)
    {
        _myconn = info;
    }
  }
```

Als letztes soll der „Catalog Mode“ erklärt werden. Hierbei wird dem Benutzer ermöglicht, Web Parts die gar nicht auf der Seite angezeigt werden zu der Seite hinzuzufügen. Wobei es hier drei verschiedene Kataloge gibt:

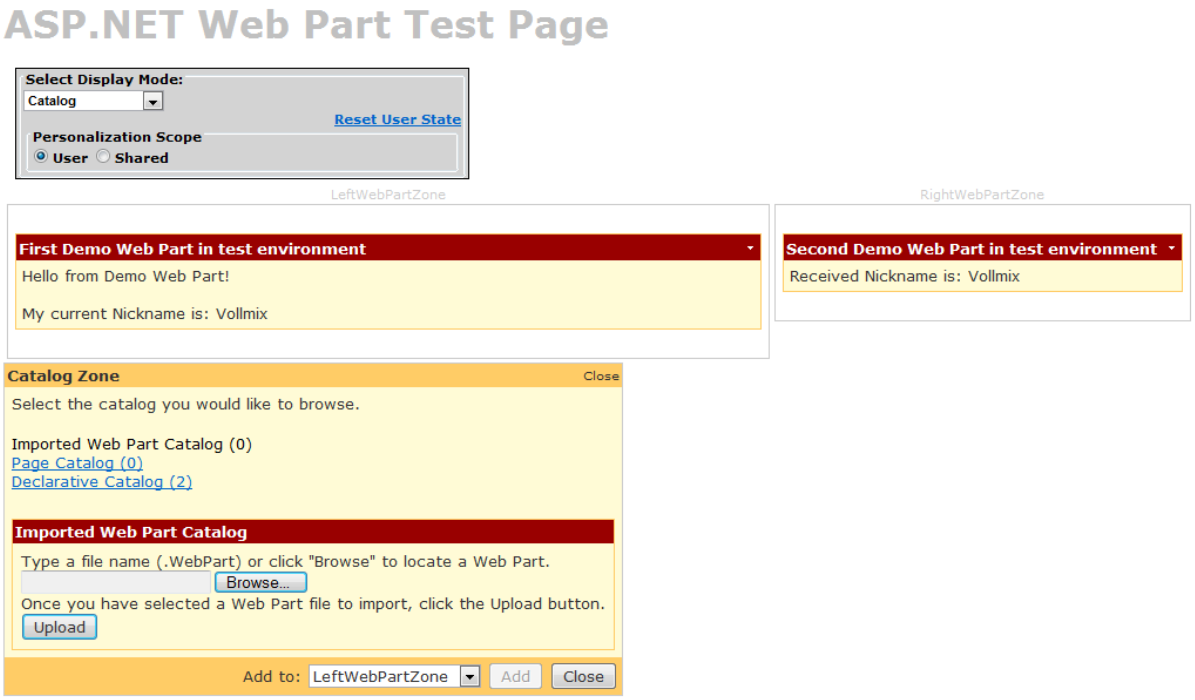

Abbildung 17: Screenshot ASP.NET Anwendung - Catalog Mode

- Den „Imported Web Part Catalog“, welcher es ermöglicht durch das laden einer *.webpart Datei, das entsprechende Web Part auf die Seite zu bekommen. Hierzu muss natürlich das entsprechende Assembly bereits im \Bin Verzeichnis der verfügbar sein.
- Im „Page Catalog“ werden die Web Parts aufgelistet die der User über den Close Menüpunkt von der Seite entfernt hat.
- Im „Declarative Catalog“ werden die Web Parts gelistet die der Entwickler direkt in der Seite (z.B.: Default.aspx) angegeben hat:

```
<asp:DeclarativeCatalogPart ID="DeclarativeCatalogPart1" runat="server">
    <WebPartsTemplate >
    <demo:ThirdDemoPart ID="DemoPart3" runat="server" Title="Third Demo
Web Part in test environment"
    Description="ASP.NET Button mit Click Count" />
    <demo:FourthDemoPart ID="DemoPart4" runat="server" Title="Fourth
Demo Web Part in test environment"
    Description="ASP.NET Label" />
    </WebPartsTemplate>
</asp:DeclarativeCatalogPart>
```

4. Hinzufügen eines neuen Web Parts

Nachdem die komplette Testumgebung konfiguriert und evaluiert ist, können wir jetzt endlich daran gehen, das Projekt füt unser selbstgeschriebenes Web Part anzulegen. Dazu im File Menü auf Add -> New Project klicken, so dass der folgende Dialog erscheint. Achtung hier sicherstellen, dass Visual C# und dann Web angeklickt ist. Dann unter My Templates den Projekttyp „ASP.NET Web Part Template“ selektieren, dann noch einen Namen eintippen und auf OK klicken.

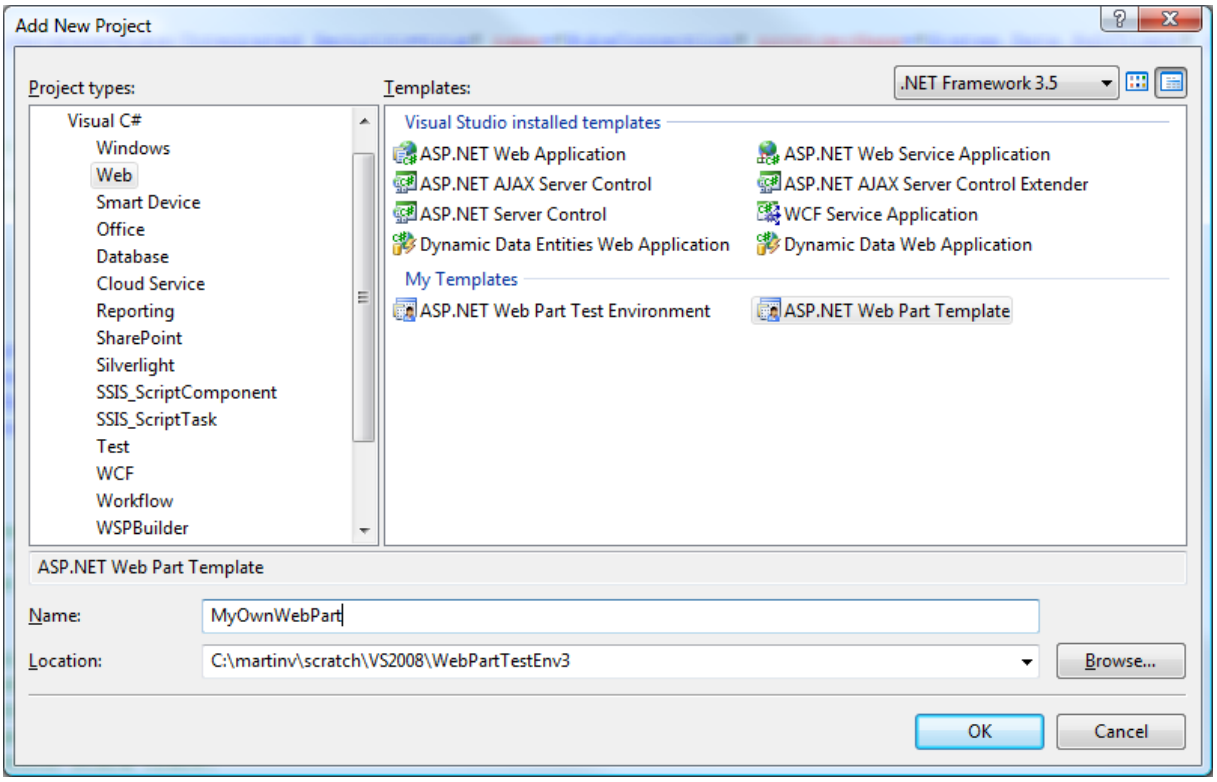

Abbildung 18: Add New Project Dialog

Damit sollte dann folgende Solution Struktur im Solution Explorer angezeigt werden:

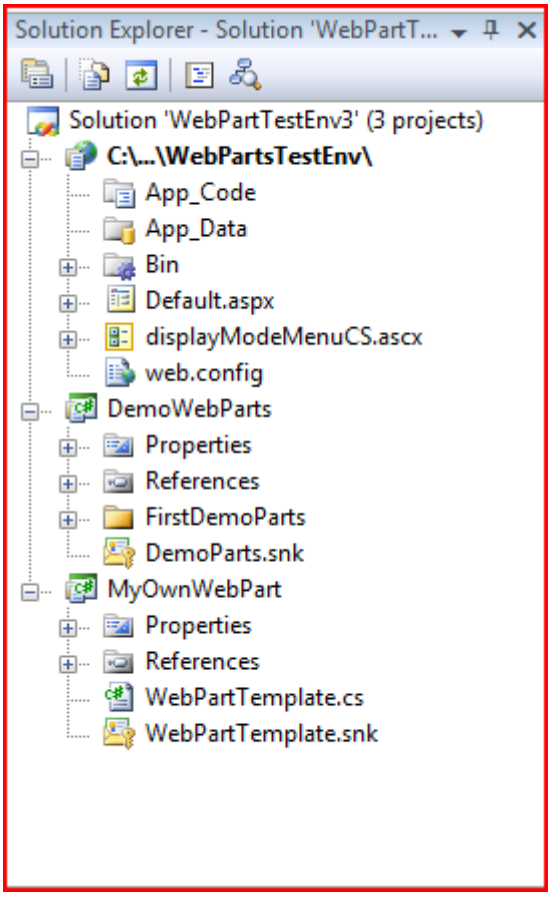

Abbildung 19: Solution Struktur

Das eingefügte Web Part ist so wie es ist schon lauffähig. Allerdings muss man das Assembly noch im Web Projekt referenzieren, so dass dieses automatisch ins \Bin Verzeichnis der Web Applikation kopiert wird. Hierzu im Solution Explorer mit der rechten Maustaste auf das Web Projekt klicken und dann mit Add Reference den entsprechenden Dialog öffnen. Am einfachsten ist es auf den Projects Tab zu klicken und dann unser „MyOwnWebPart“ auswählen.

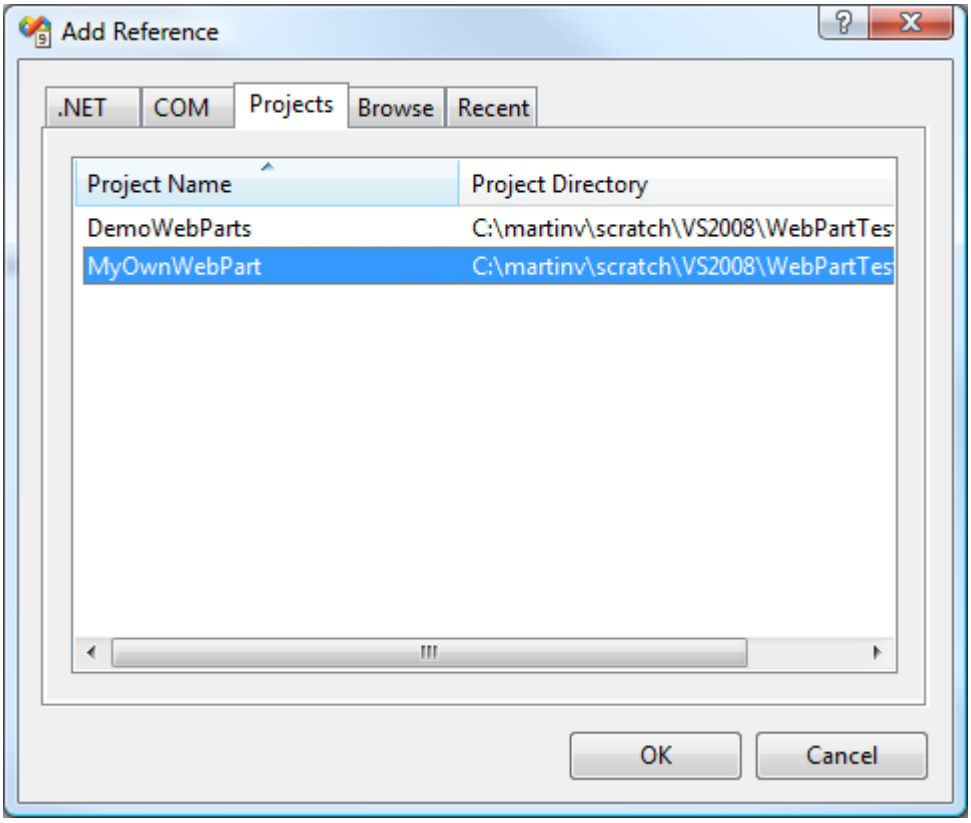

Abbildung 20: Add Reference Dialog

Um das Web Part auf Web Seite anzuzeigen muss noch die Datei Default.aspx wie folgt angepasst werden:

Zuerst muss das Assembly auf der Seite registriert werden. Diese Einträge sind am Anfang der Datei zu finden. Für unser neues Web Part kann der Eintrag dann so aussehen:

```
<%@ Register Namespace="MyOwnWebPart" Assembly="MyOwnWebPart"
TagPrefix="own" %>
```

Jetzt können wir das Web Part zu einer Web Part Zone hinzufügen:

```
<ZoneTemplate>
    <own:MyOwnWebPart ID="OwnWebPart1" runat="server" Title="Mein erstes
selbstgeschriebenes Web Part" />
    <demo:FirstDemoPart ID="DemoPart1" runat="server" Title="First Demo
Web Part in test environment" ExportMode="All" />
</ZoneTemplate>
```

5. Deployment des Web Parts auf Sharepoint

Wenn man nun das selbst erstellte Web Part auch auf einem Sharepoint Server testen möchte, dann ist es eigentlich am einfachsten das Web Part „Von Hand“ zu deployen.

1. Das Assembly muss auf den Sharepoint Server kopiert werden und zwar entweder in den „Global Assembly Cache“ oder als „private Assembly“ in das Bin Verzeichnis des Sharepoint Servers. Der Pfad bei einer Default-Installation ist:

   C:\inetpub\wwwroot\wss\VirtualDirectories\80\bin

   Wenn der Sharepoint Server im selben Netz hängt wie die Workstation und man Admin Zugriff auf die Maschine hat kann man den Kopiervorgang auch als Post-Build-Step in Visual Studio einrichten

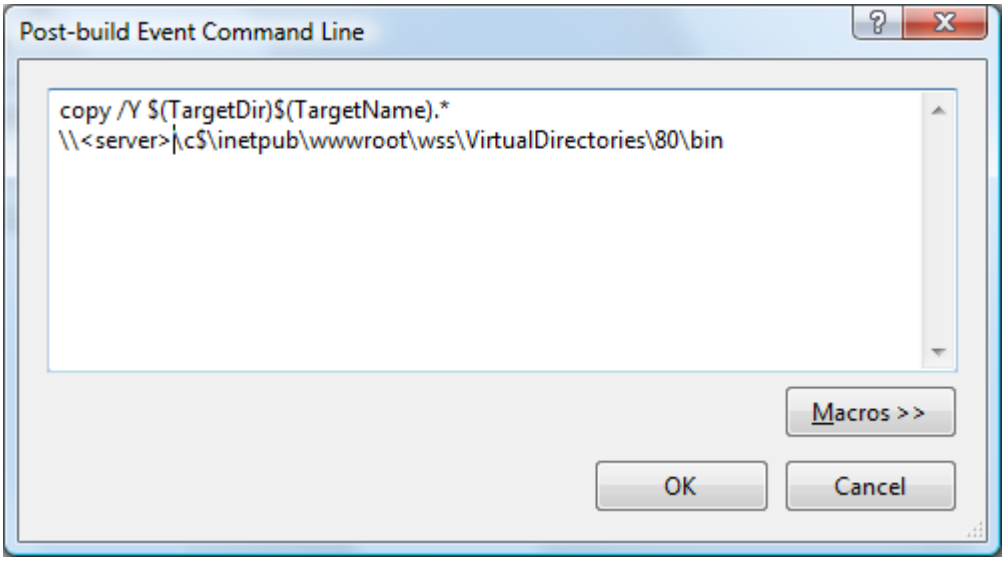

Abbildung 21: Post Build Event Command Line

2. Den Safe Control Eintrag in der Web.Config vornehmen. Ein optionaler Bestandteil dieses Eintrages ist unter anderem auch das Public Key Token. Die Frage ist nun wie ermittelt man dieses Public Key Token aus einem Assembly? Dazu kann man das Kommandozeilen Tool „sn.exe“ mit dem Parameter „-T“ verwenden. Der Auf ruf sieht dann so aus:

```
C:\>sn -T
"C:\martinv\scratch\VS2008\WebPartTestEnv3\MyOwnWebPart\bin\Debug\MyO
wnWebPArt.dll"

Microsoft (R) .NET Framework Strong Name Utility  Version 3.5.30729.1
Copyright (c) Microsoft Corporation.  All rights reserved.

Public key token is a85f470389c59b97
```

Jetzt kann der Safe Control Eintrag in der Web.Config auf dem Sharepopint Server vorgenommen werden. Die Datei web.config befindet sich übrigens auch im bin Verzeichnis des Sharepoint Servers:

```
<SafeControls>
.
.
.
<SafeControl Assembly="MyOwnWebPart, Version=1.0.0.0,
Culture=neutral, PublicKeyToken= a85f470389c59b97"
Namespace="WebParts" TypeName="*" Safe="True" />

</SafeControls>
```

3. Mit dem Browser zur Sharepoint Site navigieren.
4. Auf Site Settings klicken und dann “Modify all Site Settings” auswählen
5. Hier jetzt unter Galleries auf den Link Web Parts klicken
6. Mit dem New Button dann auf die Seite mit den neu zu veröffentlichen Web Parts navigieren:

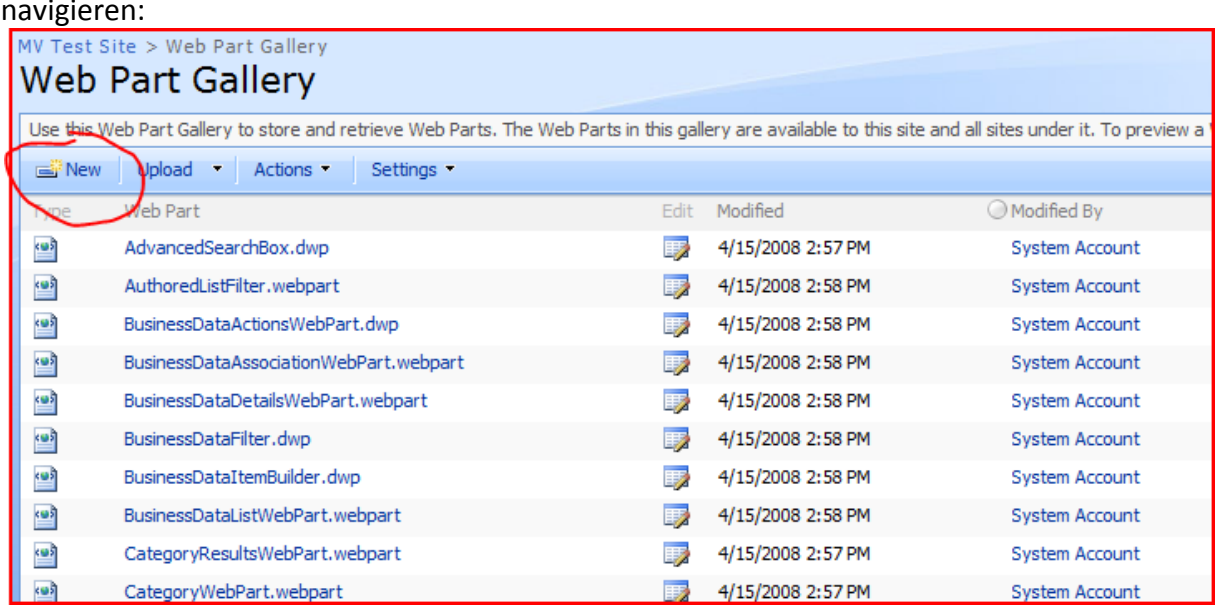

Abbildung 22: Web Part Gallery

7. Jetzt noch die Check Box vor dem zu veröffentlichenten Web Part ankreuzen und Populate klicken. Wenn das Web Part nicht in der Liste sein sollte den Safe Control Eintrag überprüfen, oder man muss den Internet Information Server restarten. Dies geht am einfachsten wenn man in einem Kommandozeilenfenster den Befehl „iisreset“ eingibt.

Abbildung 23: Populate Web Parts Seite

8. Jetzt kann man das Web Part auf jeder Seite hinzufügen:

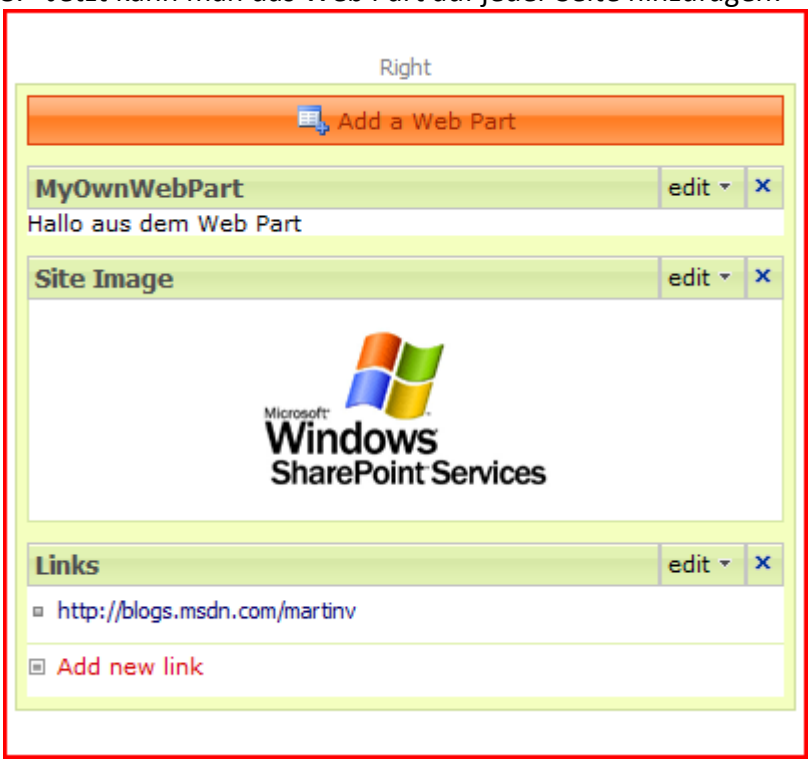

Abbildung 24: Web Part Beispiel auf einer Sharepoint Server Seite

Wenn Workstation (XP, Vista oder Windows 7) und der Server bzw. das VPC Image in derselben Domain oder Workgroup laufen, kann man Web Parts in Sharepoint auch remote mit der Workstation debuggen. Wer wissen möchte wie das geht, der sollte auf meinem Blog http://blogs.msdn.com/martinv vorbeischauen.

Viele Grüße

Martin Vollmer
Microsoft Deutschland GmbH
Developer Platform & Strategy Group

Links

SharePoint Developer Centers:
http://msdn.microsoft.com/sharepoint
http://msdn2.microsoft.com/en-us/sharepoint/

SharePoint Product Center
http://www.microsoft.com/sharepoint

Windows SharePoint Services Web Parts Resource Center
http://msdn.microsoft.com/en-us/sharepoint/bb851483.aspx

The Sharepoint Introduction for .Net Developers
http://www.microsoft.com/click/SharePointDeveloper/

Auf Codeplex (www.codeplex.com)gibt e seine Menge Tools und Templates für Sharepoint. Einfach „Sharepoint“ in der Suche eingeben.

Sharepoint Blogger

Sharepoint Team Blog
http://blogs.msdn.com/sharepoint/

Jan Tielens
http://weblogs.asp.net/jan/
Gute Infos zu WSS und Sharepoint unf auch ASP.NET Controls
Autor des SmartPart Web Parts (Container für ASP.NET user controls,
http://www.codeplex.com/smartpart)

Dan Winter
http://blogs.msdn.com/dwinter/default.aspx
Viele gute Tipps und Beispiele