

Kochen mit Patrick

Hier kommt die versprochene Fortsetzung der Tipps zu ASP.NET AJAX. Es geht darum, wie Sie anstelle nichtssagender Meldungen eigene aussagekräftige Fehlertexte erzeugen. Außerdem lesen Sie, wie Sie die Browser-History mit AJAX nutzen. Zu essen gibt's dann gefüllte Paprikaschoten.



dnpCode
A0905Kochen

Patrick A. Lorenz ist Geschäftsführer der PGK GmbH, einem auf .NET spezialisierten Technologiedienstleister. Daneben ist er als Autor tätig. Sein neuestes Buch „ASP.NET 3.5 mit AJAX“ beschreibt die Neuerungen in .NET 3.5 für Webentwickler. In seiner Freizeit ist Patrick Hobbykoch. Sie erreichen ihn unter www.pgk.de oder lorenz@pgk.de.



[Abb. 1] Der IE7 in gewohnter Entwickler-unfreundlicher Ausführlichkeit.

```
<customErrors mode="On">
  <error statusCode="500"
    redirect="/error500.aspx"/>
</customErrors>
```

Dieses Standardverhalten lässt sich mithilfe der Eigenschaft *AllowCustomErrorsRedirect* verändern. Wird der Wert vom Standard *true* auf *false* geändert, erfolgt keine Umleitung und es wird wieder die reguläre Browser-Meldung ausgegeben.

Um die clientseitige Verarbeitung von serverseitigen Fehlern selbst in die Hand zu nehmen, können Sie sich in den Lebenszyklus der Client-Klasse *PageRequestManager* einklinken, ähnlich wie ich dies beim Fortschrittsbalken in [1] gemacht habe. In diesem Fall relevant ist das *EndRequest*-Ereignis, das nach Abschluss eines asynchronen Callbacks ausgelöst wird. Aus den Ereignisargumenten lässt sich über die *Error*-Eigenschaft ein *JavaScript-Error*-Objekt ermitteln und daraus zum Beispiel die Fehlermeldung auslesen.

Das *Error*-Objekt wird nach Verarbeitung des *EndRequest*-Ereignisses regulär geworfen und führt zu der beschriebenen Fehleranzeige im Browser. Dies lässt sich verhindern, indem der Fehler mithilfe der Eigenschaft *ErrorHandled* als behandelt markiert wird. Der folgende JavaScript-Quelltext zeigt, wie ein etwaiger Fehler abgefangen und dem Browser über eine Alert-Box angezeigt wird:

```
<script>
  Sys.WebForms.PageRequestManager
    .getInstance()
    .add_endRequest(EndRequest);
```

Eigentlich schon für Heft 4 angekündigt, gibt es nun pünktlich zu Ostern die versprochene Fortsetzung der Tipps rund um das Thema ASP.NET AJAX. Den ersten Teil finden Sie in der März-Ausgabe von *dotnetpro* [1].

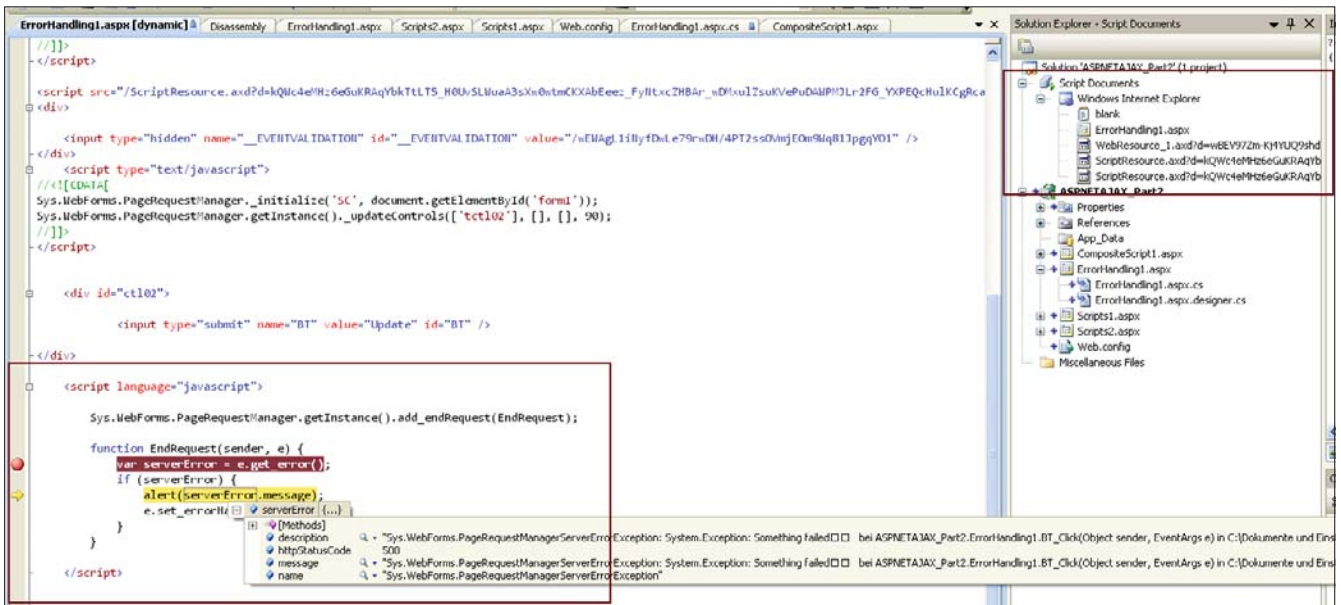
ASP.NET-AJAX-Fehlerbehandlung

Das Verhalten von ASP.NET AJAX im Falle eines serverseitigen Fehlers hängt von einer Reihe von Parametern ab. Im Zweifelsfall aber resultiert er in der browserspezifischen Behandlung von Skript-Fehlern, die nach der Konfiguration angezeigt oder auf die durch ein unscheinbares Icon hingewiesen wird. Abbildung 1 zeigt die mehr oder weniger nutzlose Meldung des Internet Explorers 7.

Von zentraler Bedeutung für die Art der Fehlerbehandlung sind folgende Aspekte:

- Der Ursprung des Aufrufs: Lokale Aufrufe auf der Entwicklungsmaschine werden meist anders behandelt als in der produktiven Umgebung.
- Die Eigenschaft *AllowCustomErrorsRedirect* des verwendeten *ScriptManager*-Controls.
- Die ASP.NET-spezifische Konfiguration der „Custom Errors“ in der *web-config*.

Das Standardverhalten sieht vor, dass die ASP.NET-spezifische Konfiguration verwendet wird. Greift der Entwickler lokal zu, wird die Fehlermeldung ähnlich wie in Abbildung 1 ohne Stack Trace ausgegeben. Andere Benutzer erhalten die Standardnachricht für den Status-Code 500, es sei denn, es ist eine Fehlerseite hierfür konfiguriert. In diesem Fall wird eine Umleitung auf die vorgegebene Seite durchgeführt:



[Abb. 3] AJAX-Debugging in Visual Studio 2008.

```
<asp:ScriptReference
  Name="MicrosoftAjax.js" />
<asp:ScriptReference
  Name="MicrosoftAjaxWebForms.js" />
</Scripts>
</CompositeScript>
</asp:ScriptManager>
```

Der Vorteil in dieser Verkettung liegt in der potenziell drastischen Reduzierung der Serverabfragen. Vergleichbar mit Bildern, werden auch Skript-Dateien einzeln vom Server übertragen und produziert damit einen gewissen Overhead. Dieser wird durch die Zusammenführung verringert und damit die Performance der Applikation gesteigert. Nachteil dieses Ansatzes ist das clientseitige Caching der JavaScript-Bibliotheken. Damit dieses überhaupt sinnvoll

greifen kann und gleiche JavaScript-Daten durch unterschiedliche Kombinationen nicht mehrfach geladen werden, sollte auf allen Webseiten möglichst die gleiche Zusammenstellung verwendet werden.

Am besten platzieren Sie das *ScriptManager*-Control aber ohnehin zentral auf einer Masterpage.

Die History mit Ajax nutzen

Als zentrale Nachteile von Ajax werden unter anderem die fehlende Unterstützung für die Browser-Historie, Favoriten und so weiter benannt – oder mit anderen Worten die Adressierung eines Seitenzustands über eine dedizierte URL. Dieses aus partiellen Seitenaktualisierungen resultierende Grundproblem teilen sich die unterschiedlichen

Ajax-Frameworks gleichermaßen. Mittlerweile gibt es einige Varianten, das Problem zu umgehen. Ein solcher Ansatz findet sich seit dem .NET 3.5 Service Pack 1 auch in ASP.NET AJAX wieder.

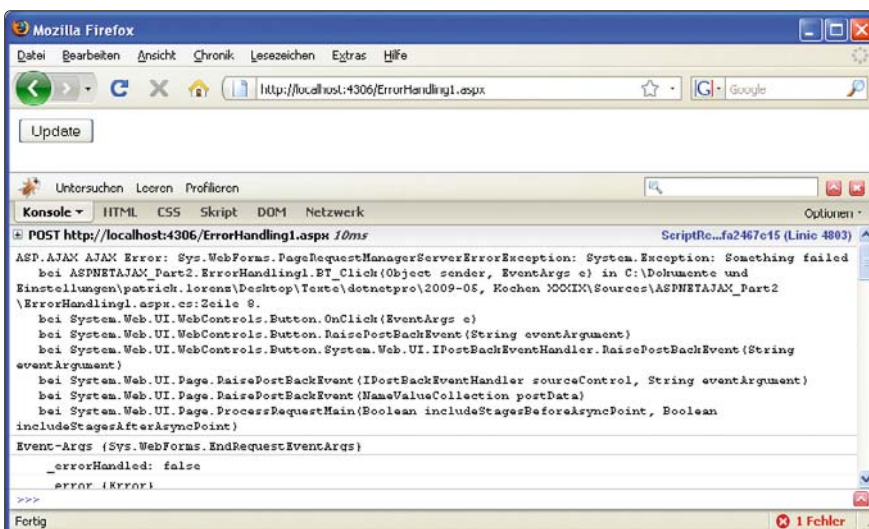
Das neue Konzept erlaubt es dem Entwickler, gezielt Einträge in der Historie des Browsers vorzunehmen und mit diesem bestimmte Zustandsinformationen zu verknüpfen. Navigiert der Benutzer innerhalb der Historie, werden die Zustandsinformationen zur Verfügung gestellt und sollen dem Aufbau des ursprünglichen Seitenzustands helfen.

Das folgende Beispiel enthält innerhalb eines *UpdatePanel*-Controls ein Eingabefeld und einen Button. Jeder Klick auf den Button speichert die aktuelle Eingabe in der Historie. Über die Navigationselemente des Browsers lässt sich zwischen den Eingaben wechseln.

```
<asp:ScriptManager id="SC"
  runat="server" EnableHistory="true"
  OnNavigate="SC_Navigate" />

<asp:UpdatePanel ID="UpdatePanel1"
  runat="server">
  <ContentTemplate>
    <asp:TextBox ID="TB" runat="server" />
    <asp:Button id="BT" runat="server"
      OnClick="BT_Click" Text="OK" />
  </ContentTemplate>
</asp:UpdatePanel>
```

Das Beispiel unterscheidet sich von anderen eigentlich nur durch die im *ScriptManager* gesetzte Eigenschaft *EnableHistory* sowie die Ereigniszuweisung für *Navigate*. Die



[Abb. 4] ASP.NET gibt Daten in der Firebug-Fehlerkonsole aus.

Hackfleisch in Gemüse

Wussten Sie, dass es bis Mitte 2007 eine eigene Hackfleischverordnung (HackfleischVO) gab? Tja, in unserer europaweit aufgeblähten Bürokratie verwundert das kaum. Die in der Verordnung enthaltenen Regeln gibt es übrigens nach wie vor. Deren zentrales Element: Abgesehen von speziell verpacktem Supermarkt-Hack darf rohes Hackfleisch ausschließlich am Tag der Herstellung in den Verkehr gebracht werden.

Gefüllte Paprika

Eine vermeintliche PaprikaVO regelt, dass dieser Klassiker immer wieder gerne auf unseren Tischen landet. Auch ich koche und esse sie sehr gerne. Die folgende Variante gart übrigens nicht im Backofen, sondern im Tomatentopf.

Kochen Sie zunächst Ihren Lieblingsreis nach Vorschrift, und zwar etwa 25 Gramm pro Person und Paprikaschote. Lassen Sie ihn anschließend abtropfen und abkühlen.

Schneiden Sie eine kleine Zwiebel in feine Würfel und dünsten Sie sie in einem sehr großen Topf in etwas Öl an. Geben Sie dann den Inhalt einer 800-Gramm-Dose Tomaten mitsamt Saft oder eine ähnliche Menge frischer, grob gewürfelte Tomaten dazu. Lassen Sie die Mischung gesalzen und gepfeffert bei schwacher Hitze köcheln.

Würfeln Sie eine weitere kleine Zwiebel sowie ein halbes Bund Petersilie und zwei bis drei Zehen Knoblauch ebenfalls fein und mischen Sie die Zutaten zusammen mit dem Hackfleisch, dem Reis, einem Ei, einem Esslöffel Senf, Salz, Pfeffer, Oregano und nach Geschmack einem Teelöffel Cayennepfeffer.

Schneiden Sie von den möglichst gleich großen Paprikaschoten – etwa eine pro Person – den Deckel ab und lösen Sie das Innenleben heraus. Füllen Sie sie mit der Mischung und geben Sie sie dann mitsamt Paprika-Deckel in den Topf. Deckel drauf, 45 Minuten leicht köcheln lassen und genießen. Mahlzeit!



eigentliche Arbeit leisten zwei Eventhandler im Code-behind der Seite:

```
protected void BT_Click(object sender,
    EventArgs e)
{
    var state = new NameValueCollection();
    state["text"] = this.TB.Text;
    this.SC.AddHistoryPoint(state,
        string.Concat("Text: ",
            this.TB.Text));
}
protected void SC_Navigate(object
    sender, HistoryEventArgs e)
{
    var state = e.State;
    this.TB.Text = state["text"];
}
```

Das erste Ereignis reagiert auf den Button-Klick. Hier wird eine neue *NameValueCollection* (String-Dictionary) instanziiert und mit dem neuen Inhalt des Eingabefeldes befüllt. Anschließend wandert die Collection zusammen mit einem Titel an die Methode *AddHistoryPoint* des *ScriptManager*-Controls. ASP.NET AJAX serialisiert die



[Abb. 5] ASP.NET AJAX speichert Seitzustände in der Historie.

se Informationen und fügt sie clientseitig dem Query-String eines neuen Eintrags in der Browser-Historie hinzu (Abbildung 5). Navigiert der Benutzer innerhalb seiner Historie, so wird das serverseitige Ereignis *Navigate* des *ScriptManager*-Controls ausgelöst. Diesem wird die aus dem Query-String deserialisierte *NameValueCollection* übergeben, und der Entwickler ist in der Lage, den ursprünglichen Seitzustand wiederherzustellen. Die Verwendung der

neuen History-Points ist sehr einfach, allerdings sind die Möglichkeiten durchaus limitiert. Wirklich viele Informationen lassen sich im Query-String nicht sinnvoll unterbringen, und ob sich damit der Seitzustand tatsächlich wiederherstellen lässt, dürfte insbesondere von der Komplexität der jeweiligen Seite abhängen. [bl]

[1] Patrick A. Lorenz, Kochen mit Patrick, dotnetpro 3/2009, Seite 115 ff.