Globale Ereignisse in ASP.NET-Webanwendungen

Die Anwendungsdatei "Global.asax"

Die Global.asax wird generell als die (globale) Anwendungsdatei bezeichnet da sie, obwohl sie nur optional ist, eine zentrale Rolle innerhalb einer ASP.NET-Anwendung spielt. Hauptaufgabe der Global.asax-Datei ist die Definition von Ereignisbehandlungsroutinen für seitenübergreifende Ereignisse, wie zum Beispiel den Start einer Sitzung oder der gesamten Webanwendung. Eine Nebenaufgabe ist die globale Instanziierung von Objekten, die auf allen Seiten verfügbar sind.

Die Global.asax muss im Wurzelverzeichnis der Anwendung definiert werden. Wird diese Datei zur Laufzeit verändert, werden alle aktuellen Sitzungen beendet und die Webanwendung wird neu initialisiert.

Die Global.asax-Datei beginnt mit der Direktive @Application. Sie kann <OBJECT>-Tags und <SCRIPT>-Blöcke mit Ereignisbehandlungsroutinen enthalten. Der Inhalt dieser Klasse wird vom ASP.NET-Arbeitsprozess in eine Klasse umgesetzt, die von System.Web.HttpApplication erbt.

Listing 5:
Beispiel für eine global.asax-Datei (Single-File-Modell)

```
<%@ Application %>


<OBJECT RUNAT=Server SCOPE=Session ID=FSO

PROGID="Scripting.FileSystemObject">

</OBJECT>


<object id="ENTRY" scope="application"

class="System.DirectoryServices.DirectoryEntry"
runat="server" />
```

```
<script runat="Server">

 Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)

  Response.Write("Session gestartet!")

 End Sub

</script>
```

Die Global.asax-Datei kann auch nach dem Code-Behind-Modell implementiert werden. In diesem Fall enthält die ASAX-Datei nicht mehr als den Verweis auf die Code-Behind-Datei und –Klasse sowie die <OBJECT>-Tags. Die Ereignisbehandlungsroutinen wandern in die Global.asax.vb. Die Klasse in der Code-Behind-Datei wird von Visual Studio .NET einfach Global genannt. Andere Namen sind aber möglich.

Tabelle 3:
Grundgerüst der Implementierung der Global.asax

Global.asax	Global.asax.vb
<%@ Application Codebehind="Global.asax.vb" Inherits="Kapitel5.Global" %>	Public Class Global Inherits System.Web.HttpApplication End Class

Hinweis

Visual Studio .NET unterstützt nur Global.asax-Dateien nach dem Code-Behind-Modell. Sie können in Visual Studio .NET nur den Quellcode der Global.asax.vb, nicht aber der Global.asax-Datei selbst bearbeiten. Dazu müssen Sie einen beliebigen Texteditor verwenden.

Globale Ereignisse

Es gibt verschiedene Arten von Ereignissen in der Global.asax-Datei. Dabei gibt es zwei Unterscheidungskritierien. Einerseits kann man die Ereignisse unterscheiden, ob sie deterministisch oder nicht deterministisch sind. Deterministische Ereignisse treten bei jeder HTTP-Anfrage bzw. der zugehörige HTTP-Antwort auf. Nicht-deterministische Ereignisse treten nur unter bestimmten Voraussetzungen, z.B. im Fehlerfall auf.

Das zweite Unterscheidungskriterium ist die Herkunft der Ereignisse. Die Global.asax-Datei bezieht Ereignisse einerseits von ihrer Oberklasse HttpApplication und andererseits aus HTTP-Modulen, die der Anwendung hinzugefügt wurden. Das folgende Listing zeigt einen Ausschnitt aus der machine.config-Datei mit den Standard-HTTP-Modulen.

Listing 6:
<httpModules> der Machine.config

```
<httpModules>

  <add name="OutputCache" type="System.Web.Caching.OutputCacheModule" />
```

```
  <add name="Session" type="System.Web.SessionState.SessionStateModule" />
  <add name="WindowsAuthentication" type="System.Web.Security.WindowsAuthenticationModule" />
  <add name="FormsAuthentication" type="System.Web.Security.FormsAuthenticationModule" />
  <add name="PassportAuthentication" type="System.Web.Security.PassportAuthenticationModule" />
  <add name="UrlAuthorization" type="System.Web.Security.UrlAuthorizationModule" />
  <add name="FileAuthorization" type="System.Web.Security.FileAuthorizationModule" />
</httpModules>
```

Ob ein Ereignis aus der Klasse HttpApplication oder einem HTTP-Modul kommt, macht insofern einen Unterschied, als dass die Ereignisse anders deklariert werden. Im ersten Fall ist der Ereignisbehandlungsroutine irrelevant; man deklariert mit dem Schlüsselwort Handles, welches Ereignis die Routine behandeln soll.

```
Private Sub Global_EndRequest(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.EndRequest

End Sub

Private Sub Global_Error(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Error

End Sub
```

Im Fall der HTTP-Modul-Ereignis, ist allein der Name wichtig.

```
Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)

End Sub

Sub FormsAuthentication_OnAuthenticate(ByVal sender As Object, ByVal e As
System.Web.Security.FormsAuthenticationEventArgs)

End Sub
```

Hinweis

Für die weitere Betrachtung soll die Herkunft der Ereignisse aber keine Rolle spielen. Wichtig ist nur zu wissen, dass durch die Installation zusätzlicher HTTP-Module auch weitere globale Ereignisse hinzukommen können.

Ereignisse aus HTTP-Modulen

Bereits im klassischen ASP gab es vier globale Anwendungs-Ereignisse:

- Application_OnStart()
- Application_OnEnd()
- Session_OnStart()
- Session_OnEnd()

Diese Ereignisse sind weiterhin vorhanden. Sie heißen jetzt aber offiziell ohne das "On" im Namen, also z.B. Session_Start() statt Session_OnStart(). Die alten Namen sind aber aus Gründen der Kompatibilität weiterhin nutzbar. Da .NET Ereignisse mit mehreren Subscribern unterstützt (sogenannte Multicast-Ereignisse), werden beide Ereignisbehandlungsroutinen ausgeführt, wenn sowohl Session_Start() als auch Session_OnStart() vorhanden ist.

Mittels dieser Ereignisse lassen sich beispielsweise Initialisierungen von Anwendungs- und Sitzungsobjekten durchführen oder Protokollierungen über die Nutzung der Anwendung durchführen.

Tipp

Wie Sie dem **Listing 5** entnehmen können, kann man innerhalb der Session_Start()-Ereignisbehandlungsroutinen Ausgaben mit Response.Write() erzeugen. Dies war im klassischen ASP nicht möglich. Auch andere Ereignisse aus der global.asax unterstützen Ausgaben an den Browser.

Listing 7:
Allgemeine Global.asax-Ereignisse

Ereignis	Beschreibung
Application_Start()	Tritt einmalig beim Start der ASP.NET-Anwendung ein.
Application_End()	Tritt einmalig beim Beenden der ASP.NET-Anwendung ein.
Session_Start()	Bei jedem Beginn einer Benutzersitzung tritt dieses Ereignis ein.
Session_End()	Nach Beendigung einer Benutzersitzung tritt dieses Ereignis ein.

Wie diese Ereignisse verwendet werden können wird im folgenden Beispiel demonstriert. Beim Anwendungsstart (Application_Start) wird eine Zahl aus einer Textdatei gelesen. Diese beinhaltet die Anzahl über die bisherigen Starts der Anwendung. Diese wird um eins erhöht und bei beim Beenden der Anwendung (Application_End) wieder in die Datei geschrieben.

```
' ////////////////////////////////////////////////

' /Kapitel05c/global.asax

' ////////////////////////////////////////////////

Imports System.IO

Imports System.Web

Imports System.Web.SessionState

Public Class Global

    Inherits System.Web.HttpApplication

…

    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)

        ' Wird ausgelöst, wenn die Anwendung gestartet wird.

        Dim _StreamReader As StreamReader
```

```
    Dim _AnzahlStarts As String
    Dim _Datei As String
    _Datei = Server.MapPath("AnzahlApplicationStarts.txt")
    Try
      _StreamReader = File.OpenText(_Datei)
      _AnzahlStarts = _StreamReader.ReadLine()
    Catch exc As Exception
      ' --- Noch keine Datei vorhanden
      _AnzahlStarts = 0
    Finally
      If Not (_StreamReader Is Nothing) Then
        _StreamReader.Close()
      End If
    End Try
    Application("AnzahlApplicationStarts") = CInt(_AnzahlStarts) + 1
End Sub

Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Wird ausgelöst, wenn die Sitzung gestartet wird.
    Application("AnzahlOffenerSessions") = Application("AnzahlOffenerSessions") + 1
    Application("AnzahlGesamtSessions") = Application("AnzahlGesamtSessions") + 1
End Sub

Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
    ' Wird ausgelöst, wenn die Sitzung beendet wird.
    Application("AnzahlOffenerSessions") = Application("AnzahlOffenerSessions") - 1
End Sub

Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
    ' Wird ausgelöst, wenn die Anwendung beendet wird.
```

```
        Dim _StreamWriter As StreamWriter

        Dim _Datei As String

        _Datei = Server.MapPath("AnzahlApplicationStarts.txt")

        Try

            ' --- Anzahl Starts in Datei schreiben

            _StreamWriter = File.CreateText(_Datei)

            _StreamWriter.WriteLine(Application("AnzahlApplicationStarts"))

        Catch exc As Exception

            ' --- Fehlerbehandlung

        Finally

            ' --- Aufräumen

            If Not (_StreamWriter Is Nothing) Then

                _StreamWriter.Close()

            End If

        End Try

    End Sub

…

End Class
```

Zusätzlich werden in diesem Beispiel alle Sessions gezählt die gerade offen sind, bzw. seit dem Start der Anwendung geöffnet wurden.

Tipp

Um das Application_Start- und Application_End-Ereignis manuell auszulösen, muss die Anwendung vollständig beendet, bzw. gestartet werden. Dies kann am leichtesten über das Beenden und Starten des WWW-Publishing-Dienstes geschehen.

Ereignisse der Klasse "HttpApplication"

Durch HttpApplication stehen Ereignisse zur Verfügung, die bei jedem einzelnen Seitenaufruf in der unten aufgeführten Reihenfolge auf. Diese können dazu verwendet werden in den Verarbeitungsprozess einer Anforderung einzugreifen, z.B. um einen Kopfzeile vor bzw. eine Fusszeile nach jeder Seite auszugeben.

Tabelle 4: ASP.NET-Anwendungs-Ereignisse (deterministisch)

Ereignis	Beschreibung
BeginRequest()	Kennzeichnet eine neu eingegangene Anforderung.
AuthenticateRequest()	Signalisiert dass die aktuelle Anfrage durch ein Sicherheitsmodul authentifiziert wurde.
AuthorizeRequest()	Signalisiert, dass die aktuelle Anfrage durch

	ein Sicherheitsmodul autorisiert wurde.
ResolveRequestCache()	Erlaubt eine Anfrage aus dem Cache. Wird vom Output Cache Modul verwendet um die Anforderung eventuell beschleunigt zu verarbeiten.
AcquireRequestState()	Signalisiert, dass eine Anforderung auf den Status der Anwendung (z.B. Sitzungsstatus) verwendet, bzw. diesen benötigt.
PreRequestHandlerExecute()	Ereignis, das vor der Ausführung des eigentlichen Anwendungsaufruf auftritt.
PostRequestHandlerExecute()	Signalisiert, dass der HTTP-Handler die Anfrage vollständig verarbeitet hat.
ReleaseRequestState()	Signalisiert, dass der Anforderungsstatus gespeichert werden soll, weil die Anwendung an dieser Stelle die Anforderung beendet. Dieses Ereignis weißt die Status Module an die aktuellen Statusdaten zu speichern.
UpdateRequestCache()	Kennzeichnet die aktuelle Anforderung als vollständig verarbeitet und bereit für die Transferierung in den ASP.NET Cache.
EndRequest()	Signalisiert, dass die gesamte Verarbeitung beendet ist.

Diese aufgeführten Ereignisse stehen der gesamten Anwendung über die Global.asax zur Verfügung.

Beispiel

Im folgenden Beispiel wird das BeginRequest- und EndRequest-Ereignis dazu verwendet um vor und nach dem eigentlichen Generieren der Webseite zusätzliche Text auszugeben.

```
' ////////////////////////////////////////////////////////////////////

' /Kapitel05c/global.asax

' ////////////////////////////////////////////////////////////////////

Imports System.IO

Imports System.Web

Imports System.Web.SessionState

Public Class Global

    Inherits System.Web.HttpApplication

…

    Private Sub Global_BeginRequest(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.BeginRequest
```

```
        Response.Write("-- Überschrift --<br>")

    End Sub

    Private Sub Global_EndRequest(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.EndRequest

        Response.Write("-- Copyright'2002 --<br>")

    End Sub

…

End Class
```

Nicht-deterministische HttpApplication-Ereignisse

Zusätzlich gibt es HttpApplication-Ereignisse die nicht immer und nicht in einer bestimmten Phase der Anwendung auftreten. Diese Ereignisse werden in der folgenden Tabelle aufgeführt. Diese nicht-deterministischen Ereignisse treten nur unter den beschriebenen Bedingungen auf.

Tabelle 5:
nicht
deterministisch
Anwendungs-
Ereignisse

Ereignis	Beschreibung
PreSendRequestHeaders()	Tritt vor dem Senden von HTTP-Header ein. Dadurch wird es ermöglicht, den Header zu manipulieren.
PreSendRequestContent()	Tritt kurz vor dem Senden von Daten an den Client auf.
Error()	Signalisiert eine unbehandelte Ausnahme.

Die auftretenden Ereignisse beim Aufruf einer Webseite bieten sehr viele Ansatzpunkte um eigene Verarbeitungslogik zu integrieren. Diese zusätzlichen Funktionalitäten können einerseits in der eigentlichen Global.asax-Datei oder ihrer Code-Behind-Klasse, behandelt werden.

```
' ////////////////////////////////////////////////////////////////////

' /Kapitel05c/global.asax

' ////////////////////////////////////////////////////////////////////

Imports System.IO

Imports System.Web

Imports System.Web.SessionState

Public Class Global

    Inherits System.Web.HttpApplication
```

```
...
    Private Sub Global_Error(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Error
        Dim _LetzterFehler As Exception
        _LetzterFehler = Server.GetLastError()
        Response.Write("<hr>" & _LetzterFehler.Message & "<hr>")
        Server.ClearError()
    End Sub
...
End Class
```

Dieses Beispiel behandelt aufgetretene Fehler und umgeht durch Server.ClearError() die Standard-Fehlermeldung von ASP.NET.

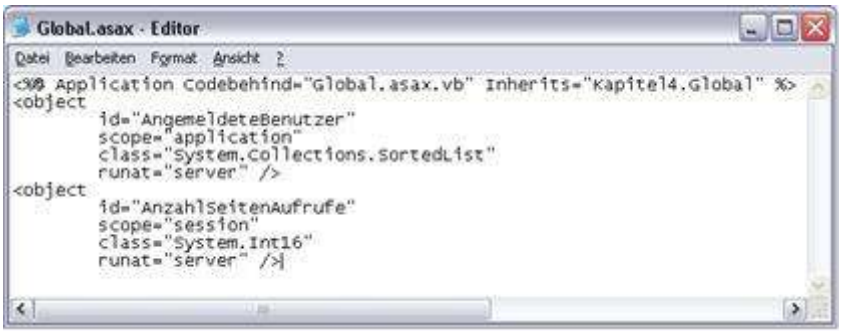
Statische Objekte innerhalb der Anwendung oder Sitzung

Über die Global.asax lassen sich zusätzlich Instanzen von statischen Objekten erzeugen, die von jedem Webform innerhalb der gesamten Anwendung verwendet werden können. Dadurch erhält man zentral die Möglichkeit globale Objekte zu definieren, die immer zur Verfügung stehen. Die Definition geschieht über das <OBJECT>-Tag im HTML-Code der Global.asax (nicht in der Code-Behind-Datei!), außerhalb des <SCRIPT>-Blocks.

Hinweis

Um den eigentlichen Quelltext (nicht die Code-Behind-Klasse) der Global.asax zu bearbeiten, muss man auf einen einfachen Texteditor zurückgreifen. Visual Studio .NET unterbindet diesen direkten Zugriff auf die Global.asax.

Abbildung **4**:
Statische Objekte in der Global.asax



Diese statischen Objekte haben einen Gültigkeitsbereich (den so genannten Scope) in dem sie verwendet werden können. Dieser Scope kann sich auf eine Benutzersitzung (SCOPE="Session") oder Anwendungsweit über alle Sitzungen hinweg erstrecken (SCOPE="Application").

Um auf die entsprechenden Objekte zur Laufzeit zugreifen zu können kann die Application.StaticObjects-Liste bzw. die Session.StaticObjects-Liste verwendet werden.

```
' ////////////////////////////////////////////////////////
```

```
' /Kapitel05/StatischeObjekte.asp.vb

' ////////////////////////////////////////////////////////

Dim EinStatischesObjekt As DictionaryEntry

For Each EinStatischesObjekt In Application.StaticObjects

    C_ApplikationObjekte.Text += EinStatischesObjekt.Key

    C_ApplikationObjekte.Text += " - "

    C_ApplikationObjekte.Text += EinStatischesObjekt.Value.GetType.ToString()

    C_ApplikationObjekte.Text += "<br>"

Next

For Each EinStatischesObjekt In Session.StaticObjects

    C_SessionObjekte.Text += EinStatischesObjekt.Key

    C_SessionObjekte.Text += " - "

    C_SessionObjekte.Text += EinStatischesObjekt.Value.GetType.ToString()

    C_SessionObjekte.Text += "<br>"

Next
```

Das Resultat sieht man in der folgenden Abbildung.

Abbildung **5**:
Zugriff auf statische Objekte zur Laufzeit