12 Entwicklung von dynamischen Webseiten

Der gängige Name für dynamische Webseiten ist Server Pages. Dabei handelt es sich um Webseiten, die an einigen Stellen Programmcode eingebaut haben, um den Inhalt der Seite zu bestimmen. Die wichtigsten Technologien sind dabei das traditionelle CGI, ASP für Microsofts IIS, JSP für die Java-Webserver wie WebSphere oder Apache Tomcat und BSP für den SAP Web AS.

12.1 Server Pages und Scripting

Schnittstellen zwischen Webseiten und ausführbaren Anwendungen

Server Pages sind Dateien auf einem Webserver, die sich mit einer URL ansprechen lassen und ausführbaren Programmcode enthalten. Dieser Programmcode gibt gegebenenfalls einen Text zurück, der dann an die Stelle des Programmcodes in die Seite eingesetzt wird. Die so erzeugte Seite wird dann als Ergebnis der URL-Anfrage an den Requester zurückgegeben.

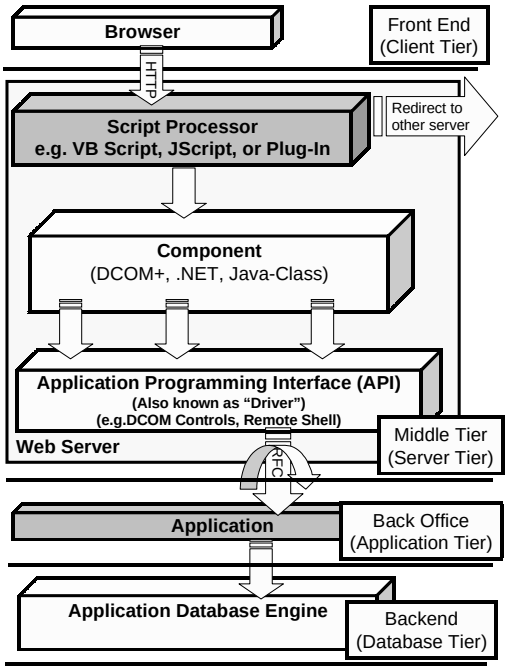

Abbildung 12.1 Ein Browser ruft ein Skript und das Skript ruft eine Methode

Das Funktionsprinzip ist dabei bei allen Webservern ähnlich. Die Server Page wird vom Webserver an einen Script-Prozessor übergeben, der den Programmcode evaluiert, ausführt und den Ergebnisstring in die Server Page einfügt.

12.1.1 Common Gateway Interface – CGI

Das *Common Gateway Interface* (CGI) ist die älteste Implementierung für dynamische Webseiten. Dabei kann der Requester eine URI angeben, die auf ein ausführbares Programm auf dem Webserver verweist. Zusätzlich können Parameter an die URI übergeben werden. CGI prüft, ob der Requester berechtigt ist, das Programm auszuführen.

Folgendes ist ein Beispiel eines Aufrufs einer DLL via CGI:

```
HTTP.//www.logosworld.com/CGI/tes-
tiis.dll?method="test"&myname="Goofy"
```

12.1.2 Active Server Pages – ASP und ASP.NET

Scripting wird von VBS oder Microsoft.NET ausgeführt

Active Server Pages (ASP) ist die Microsoft-Variante von Server Pages für den Microsoft IIS. ASP.NET ist der Nachfolger von ASP und baut auf dem Microsoft.NET-Framework auf. Während ASP nur Visual Basic Script und JScript unterstützt, erlaubt ASP.NET alle Microsoft.NET-Programmiersprachen. ASP-Seiten haben gewöhnlich die Dateierweiterung *.asp* und ASP.NET erkennt man an der Extension *.aspx*. In der ASP-Variante wird immer der VBS-Script-Prozessor *WScript* aufgerufen. Dieser ist in der Lage, jedes beliebige DCOM-Objekt anzusprechen, also auch ein beliebiges Windows-Programm auszuführen.

Im nächsten, einfachen Beispiel versuchen wir, Verbindung mit einem Webserver aufzunehmen. Die aufgerufene Seite meldet bei Erfolg die Versionsnummer des Webservers zurück.

Listing 12.1 ASP-Script, das die Versionsnummer des Webservers ermittelt

```
<html><body>
Server Side ASP Visual Basic Script
<BR>Message:
<% = "<b>Hello from <i>IIS Server</i> version "
+ server.HTTPdjsVersion) % +"</b>">
<BR>End of Message </body></html>
```

Listing 12.2 Ergebnis nach der Verarbeitung

```
<html><body>
Server Side ASP Visual Basic Script
<BR>Message:
<b>Hello from <i>IIS Server</i> version 2.0</b>
<BR>End of Message </body></html>
```

Listing 12.3 Anzeige im Browser

```
Server Side ASP Visual Basic Script
Message: Hello from IIS Server Version=2.0
End of Message
```

Durch die Verwendung von Microsoft.NET in ASP.NET ist ASP mittlerweile hinsichtlich Transaktionsfähigkeit, Stabilität und Sicherheit vollständig auf dem Niveau von Java und JSP angelangt. JSP-Code lässt sich häufig ohne Änderung unter ASP.NET ausführen, da der J#-Prozessor von Microsoft.NET weitgehend kompatibel zu Java ist.

12.1.3 Java Server Pages – JSP

Dynamische Webseiten mit Java-Code

Java Server Pages (JSP) funktionieren fast identisch wie ASP, mit dem Unterschied, dass die Programmbefehle in Java gehalten sind. Ergänzend zu ASP lassen sich die Befehle für JSP in zwei Syntax-Varianten angeben, einmal in klassischer Form, eingeschlossen in `<% ... %>`, und in XML-Syntax:

```
<%= Java Expression %>
```

oder

```
<jsp:expression>
       Java Expression
</jsp:expression>
```

Das Ergebnis einer Expression kann genau wie bei ASP und, wie wir unten sehen werden, BSP entweder direkt in den Output umgeleitet werden oder durch Zugriff auf das entsprechende Objekt:

```
Current time: <%= new java.util.Date() %>
```

oder

```
<% out.response request.getRemoteHost() %>
```

Listing 12.4 zeigt ein Beispiel einer kompletten JSP.

Listing 12.4 Beispiel einer Java Server Page

```
<!—Name of this page: login.jsp (= Target for Submit) --
>
<html>
  <head>
    <title>JSP-Beispiel</title>
  </head>

  <body>

  <%@ page import = "java.util.*" %>
  <%
    if (request.getParameter("submit") != null)
    { out.println("Ihr Username: " +
    request.getParameter("ui") + "<BR>");
      out.println("<hr>");
      out.println("Aktuelles Systemdatum/Systemzeit: "
      + new Date());  }
    else {
  %>

<HR>
<h3>Login</h3>
<form method="POST" action="login.jsp" class="shopform">
<font size="4" color=red>
</font>

<p>Please enter your User Data below to proceed:
(Userid=logos Password=world)</p>
  <table border="0" cellpadding="0" cellspacing="0"
  width="300">
    <tr><td width="175">User Name</td>
        <td width="125"><input name="ui"
        size="20"></td>
    </tr> <tr>
        <td width="175">Password</td>
        <td width="125"><input name="pw" size="20"
        type="password"></td>
```

```
    </tr>
  </table>
<hr>
  <input type="submit" value="OK" name="OK"
  tabindex="1" class="button">
  <input type="reset" value="Cancel" name="Cancel"
  tabindex="2" class="button">
</form>
<hr>

  <% } //end if
  %>

  </body>
  </html>
```

Vordefinierte Objekte in Server Pages

Es gibt eine Anzahl von vordefinierte Objekten, die jeder JSP automatisch zur Verfügung stehen. Die meisten dieser Objekte gibt es auch für ASP oder BSP. Tabelle 1.1 listet diese Objekte auf.

Objekt	Beschreibung
`request`	Zugriff auf das HTTP Request-Objekt
`response`	Container für die Antwort an den Client
`out`	Objekt zum formatierten Ausgeben von Daten auf die erzeugte JSP
`session`	Informationen der HTTP-Session
`application`	Globales Objekt zum persistenten Abspeichern von einfachen Daten
`config`	Konfigurationsdaten für die aufgerufene Seite
`pageContext`	Kontextinformationen für Java Beans (Transaktionen in Java)
`page`	Leitet ein Pragma (Direktiven für den JSP-Interpreter) ein, welches Hinweise gibt, wie die Seite zu bearbeiten ist oder welche Meta-Kommandos ausgeführt werden sollen. Beispiel »page include« zum Bekanntmachen einer Java-Bibliothek.

Tabelle 12.1 Vordefinierte Objekte von Server Pages

12.1.4 Java Servlets

Aufruf über feste Methoden

Alternativ zu JSP gibt es auch die Möglichkeit, Methoden von Java-Klassen direkt aufzurufen. Der Einstieg in die Klasse erfolgt aber immer über Methoden, deren Namen fest vorgegeben sind.

Java-Klassen, die HTTP-Antwort zurückgeben

Java ist eine Sprache, die einen großen Teil der Konventionen in die Bezeichner einzelner Objekte und Klassen hineinlegt. Dieses Prinzip wird auch bei der Implementierung eines zu CGI kompatiblen Interfaces angewendet. Wenn ein für Java ausgelegter Webserver wie Tomcat oder WebSphere einen HTTP-Request für ein Servlet empfängt, ruft der Server die passende Methode auf. Der Name der Methode wird dabei aus dem HTTP-Kommando (`GET`, `POST`, `PUT`, `HEADER`, `DELETE`) gebildet, dem das Wort `do` vorangestellt wird:

- `doGet`
  `doGet()` wird immer dann aufgerufen, wenn der Server einen HTTP GET-Request empfängt.
- `doPost`
  `doPost()` wird immer dann aufgerufen, wenn der Server einen HTTP POST-Request empfängt.
- `doXxx`
  `doXxx()` wird immer dann aufgerufen, wenn der Server einen anderen HTTP xxx-Request empfängt, wobei für xxx der Name des HTTP-Kommandos gesetzt wird.

Es gibt eine Reihe solcher HTTP-Kommandos, zum Beispiel `HEADER` oder `DELETE`, die vom Servlet optional unterstützt werden können. Werden die Methoden trotzdem aufgerufen, ohne dass sie existieren, wird eine entsprechende HTTP-Meldung zurückgesendet.

12.1.5 Business Server Pages (BSP)

Server Pages mit ABAP

Business Server Pages (BSP) sind die Scripting-Seiten des SAP Web Application Servers (Web AS). Der SAP Web AS unterstützt zwei Personalities, die ABAP Personality und die Java Personality, Sie können also wahlweise in Java oder in ABAP programmieren. Daraus darf man aber allen Gerüchten zum Trotz nicht folgern, dass es Pläne gäbe, ABAP irgendwann zu Gunsten von Java abzuschaffen. Business Server Pages werden in ABAP programmiert, ein Beispiel zeigt das folgende Listing:

Listing 12.5 Beispiel einer BSP »Hello World«

```
<%@page language="abap"%>
<html>
  <head>
    <link rel="stylesheet"
    href="../../sap/public/bc/bsp/styles/sapbsp.css">
    <title> Hello World BSP Page </title>
```

```
  </head>

  <body class="bspBody1">
    <H1>Hello World from Your BSP</H1>
    Show me current exchange rates with
    <a href="currencies.htm">
    BAPI BAPI_EXCHRATE_GETCURRENTRATES</a>
  </body>
</html>
```

12.2 Web Pages mit Active Server Pages

Das Entwickeln von dynamischen Webseiten lässt sich am besten am Beispiel von Microsofts ASP oder ASP.NET erläutern. Zum einen liegt es daran, dass VB-Programme sich auf jedem Windows-PC einfach entwickeln lassen. Gut geeignet sind hierfür die VBA-Erweiterungen von Microsoft Office, die die Möglichkeit bieten, gleichzeitig in Microsoft Word zu entwickeln, zu testen und zu dokumentieren. Hinzu kommt die wenig intuitive Syntax von Java. Visual Basic, ABAP oder Delphi sind einfacher zu erlernen.

12.2.1 ASP »Hello World«

Sniplets sind kleine Codesequenzen

Im Folgenden sehen Sie ein erstes Beispiel für ein Scriptlet in ASP, es heißt natürlich `HELLO WORLD`. Die Begriffe *Scriptlet* und *Sniplet* sind gängige Bezeichnungen für kleine Scripts.

In dem Beispiel handelt es sich um ein einfaches HTML-Dokument, in dem eine einzige Programmzeile eingefügt ist. Programmzeilen werden zwischen öffnenden und schließenden Klammern der Form `<%` und `%>` eingefügt. Das Zeichen »=« innerhalb einer Programmsequenz ist identisch mit der Methode `response.write` und bewirkt die Ausgabe des nachfolgenden Strings, der auch durch einen Ausdruck erzeugt werden kann.

Listing 12.6 Hello World mit einem ASP-Sniplet

```
<html>
  <head>
    <title>Hello World ASP SCRIPT</title>
  </head>
  <body>
    <h1>Server Side ASP Visual Basic Script</h1>
```

```
      <p>Now there should be a message from the
      Server
      <BR>Message:
      <% = "<B><U>Hello from ASP Server</U></B>" %>
      <BR>The message should be one line above</p>
      <BR>The current time is:</p>
    <% = "<B><U>"+Str(Now())+"</U></B>" %>
  </body>
</html>
```

Das nachfolgende Beispiel greift auf die COM-Komponente `Scripting.FileSystemObject` zu und demonstriert, wie beliebige COM-Objekte von ASP angesprochen werden können.

Listing 12.7 Script zur Anzeige der gefundenen Dateinamen

```
<%
Set fs = CreateObject("Scripting.FileSystemObject")
Set subdir = fs.GetFolder(".")
ii = 0
response.write subdir.path
For Each xfile In subdir.Files
ii = ii + 1
response.write "<P>"
response.write ii
response.write ": " + xfile.Name + "</P>"
Next
%>
```

12.2.2 Senden von HTML

HTML-Dokumente mit Programmstatements

An die Stelle des ASP-Codes kann mit der Methode `response.write` ein beliebiger Text, also auch HTML-Tags, eingefügt werden. Im nachstehenden Beispiel erzeugen wir eine saubere HTML-Tabelle mit ASP. Das Ergebnis sehen Sie in Abbildung 12.2.

Listing 12.8 ASP, die eine HTML-Tabelle erzeugt

```
<body>Server Side ASP Sending an HTML table
<table border="1"cellpadding="0" cellspacing="0"
width="100%">
<%
response.write "<TR><TD>A table line</TD></TR>"
```

```
response.write "<TR><TD>A table line</TD></TR>"
response.write "<TR><TD>A table line</TD></TR>"
response.write "<TR><TD>A table line</TD></TR>"
response.write "<TR><TD>A table line</TD></TR>"
%>
</table>
</body>
```

Abbildung 12.2 Ergebnis der Verarbeitung der ASP

12.2.3 Aufrufe beliebiger Windows-Objekte aus VBS

VB zu COM-Brücke

Es ist sinnvoll, komplexe Aufgaben als COM-Objekte zu realisieren. Dann spielt es auch keine große Rolle mehr, in welcher Programmiersprache die Anwendung selbst realisiert ist. Die kompilierte Applikation steht als COM-Objekt zur Verfügung und kann dann mit dem Visual Basic-Befehl `CreateObject` angesprochen werden.

Ein Beispiel, wie eine komplette Shopping-Cart-Lösung in einer einzelnen DLL verpackt werden kann, um sie dann von ASP aus anzusprechen, zeigt `IIScart2000` von IISCart (*www.iiscart.com*).

12.2.4 4.2.4BAPI-Aufruf von ASP

BAPIs werden wie alle RFC-Bausteine aufgerufen

Auch der Aufruf von SAP-BAPIs aus einer ASP-Seite heraus erfolgt durch die COM-Brücke von Visual Basic Script. Das folgende Beispiel ruft den bekannten Funktionsbaustein `BAPI_EXCHRATE_GETCURRENTRATES` auf und gibt die komplette Tabelle als HTML-Tabelle aus.

Listing 12.9 ASP zum Aufruf eines BAPIs in SAP R/3

```
<html>
<head>
<title>Example calling BAPI
```

```
BAPI_EXCHRATE_GETCURRENTRATES</title>
<link rel="stylesheet" type="text/css" href="style.css">
<%
Private Functions 'As SAPFunctionsOCX.SAPFunctions
Private LogonControl 'As SAPLogonCtrl.SAPLogonControl
Private R3Connection 'As SAPLogonCtrl.Connection

Dim Func 'As SAPFunctionsOCX.Function
Private iDATE  'As SAPFunctionsOCX.Parameter
Private tEXCH_RATE_LIST  'As SAPTableFactoryCtrl.Table

Public Sub Main()
    Dim ix 'As Integer
    Dim retcd 'As Boolean
    Dim SilentLogon 'As Boolean
    Set LogonControl =
    CreateObject("SAP.LogonControl.1")
    Set Functions = CreateObject("SAP.Functions")
    Set R3Connection = LogonControl.NewConnection
    R3Connection.Client = "000"
    R3Connection.ApplicationServer = "192.168.69.111"
    R3Connection.Language = "EN"
    R3Connection.User = "DEVELOPER"
    R3Connection.Password = "lagunas"
    R3Connection.System = "WAS"
    R3Connection.SystemID = ""
    R3Connection.SystemNumber = "18"
    R3Connection.UseSAPLogonIni = False
    SilentLogon = True

    retcd = R3Connection.Logon(0, SilentLogon)
    If retcd <> True Then MsgBox "Logon failed": Exit
    Sub
    Functions.Connection = R3Connection

    Set Func =
    Functions.Add("BAPI_EXCHRATE_GETCURRENTRATES")
    Set iDATE = Func.Exports("DATE")
    Set tEXCH_RATE_LIST = Func.Tables("EXCH_RATE_LIST")
    iDATE.Value = "20030101"
```

```
    Func.Call
    Call BAPI_to_ASP(tEXCH_RATE_LIST)
    R3Connection.logoff
End Sub

Private Sub BAPI_to_ASP(ByRef SAPtable)
    Dim iRow 'As Integer
    Dim iCol 'As Integer
    response.write"<TH>"
    For iCol = 1 To SAPtable.ColumnCount
        response.write "<TD>"
        response.write SAPtable.ColumnName(iCol)
        response.write "</TD>"
    Next
    response.write "</TH>"
    For iRow = 1 To SAPtable.RowCount
    response.write "<TR>"
    For iCol = 1 To SAPtable.ColumnCount
        response.write "<TD>"
        response.write SAPtable.Cell(iRow, iCol)
        response.write "</TD>"
    Next
    response.write "</TR>"
    Next
End Sub
%>
</head>
<body>
Server Side ASP Sending an HTML table
<table border="1" cellpadding="0" cellspacing="0"
width="100%">
<%
       Call Main
%>
</table>
</body>
</html>
```

12.2.5 Microsoft.NET Connector

.NET Connector basiert auch auf librfc32.dll

Für die Entwicklung von Schnittstellen zwischen Microsoft.NET und SAP R/3 steht ein spezieller Connector zur Verfügung. Dieser Connector basiert aber genauso wie die ActiveX-Controls auf der `librfc32.dll` von SAP. Vermutlich versucht man, den .NET-Connector etwas näher an die Aufrufkonventionen des Java-Connectors heranzuführen. Allerdings erledigt auch die COM-Brücke diese Aufgabe hervorragend, insbesondere wenn man bedenkt, dass auch der Java-Connector auf Windows-Systemen nur über das *Java Native Interface* (JNI) auf die `librfc32.dll` zugreift.

12.2.6 Parameter aus der URI abfragen

Mit jeder URI können Parameter mitgegeben werden. Bei einem `HTTP GET`-Request werden diese nach der Konvention von CGI an die URL, durch ein Fragezeichen (»?«) getrennt, angehängt. Den zusätzlichen Teil nach der URL nennt man auch einen *Querystring*. Bei einem `HTTP POST`-Request werden sie im Bauch des HTTP-Requests als Datencontainer mitverschickt.

Das Folgende ist ein Beispiel eines Querystrings als Teil einer URI:

```
http://localhost/queryst-
ring.asp?Name="Micky"&City="Duckburg"
```

Der Querystring ist `Name="Micky"&City="Duckburg"`

Für die Verwendung von Querystrings gelten folgende Regeln:

- Ein Querystring wird durch ein Fragezeichen (»?«) von der URL getrennt.
- Mehrere Parameter werden durch ein Ampersand (»&«) voneinander getrennt.

Das nachstehende Listing zeigt ein kleines ASP-Script, das den Querystring auswertet und dessen Werte anzeigt.

Listing 12.10 ASP zur Anzeige der angegebenen Querystring-Parameter

```
<HTML><BODY>
<H3>Here are the parameters which have been passed.</H3>
<P>
<% = "Querystring="+Request.QueryString %></P>
```

```
<P><% = "Input field Name is="+Request.QuerySt-
ring("NAME") %></P>
<P><% = "Input field City is="+Request.QuerySt-
ring("CITY") %></P>
</BODY></HTML>
```

In einem HTTP POST-Request werden die Daten über das Protokoll mit transportiert. Auf einer HTML-Seite werden diese Werte durch ein HTML-<FORM> erfasst:

Listing 12.11 ASP zur Anzeige der Werte von HTML-<FORM>-Variablen

```
<HTML><BODY>
<H3>Here are the parameters which have been passed.</H3>
<P>
<% = "Querystring="+Request.QueryString %></P>
<P><% = "Input field Name is="+Request.QuerySt-
ring("NAME") %></P>
<P><% = "Input field City is="+Request.QuerySt-
ring("CITY") %></P>
<P>You can address the parameters with a numeric posi-
tion index as well</P>
<P>
<% if Request.QueryString <> "" then _
resstr = "FirstParam is="+Request.QueryString(1)" %>
<% response.write resstr %>
</P>
</BODY></HTML>
```

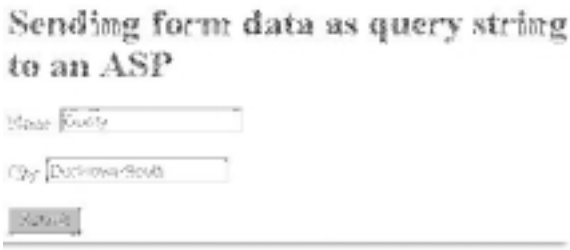

Abbildung 12.3 HTML-Seite zum Eingeben der Werte in ein <FORM>-Element

Werden die Daten eines <FORM> mit der HTTP GET-Methode versandt, erzeugt HTML einen Querystring, der aus den <FORM>-Daten gebildet wird. Im folgenden Listing sehen Sie einen HTTP GET-Request, der die Parameter als Teil der URI erhält.

Listing 12.12 <FORM> für ein HTTP GET und der erzeugte Querystring

```
<FORM METHOD="GET" ACTION="querystring.asp">
  <H1>Sending form data as query string to an ASP</H1>
  <P>Name: <INPUT TYPE="TEXT" NAME="NAME"></P>
  <P>City: <INPUT TYPE="TEXT" NAME="CITY"></P>
  <P><INPUT TYPE="SUBMIT" NAME="Submit"
  VALUE="Submit"></P>
</FORM>
```

Die Browser bauen die URI selbst nach CGI-Konvention zusammen, indem die Werte der `<FORM>`-Felder an die URL angehängt werden:

```
HTTP.//querystring.asp?NAME="Micky"&CITY="Duckburg"
```

Als nächstes folgt noch ein Beispiel für ein HTML-Formular, das beim Absenden die `<FORM>`-Daten in den Body des HTTP-Requests verpackt, anstatt sie an die URI anzuhängen.

Listing 12.13 <FORM> für METHOD=POST, das die Daten im Bauch des HTTP-Requests versendet

```
<FORM METHOD="POST" ACTION="querystring.asp">
  <H1>Sending form data as query string to an ASP</H1>
  <P>Name: <INPUT TYPE="TEXT" NAME="NAME"></P>
  <P>City: <INPUT TYPE="TEXT" NAME="CITY"></P>
  <P><INPUT TYPE="SUBMIT" NAME="Submit"
  VALUE="Submit"></P>
</FORM>
```

Nachstehend sehen Sie eine ASP-Seite, die die Werte eines `HTTP-Post` auswertet und in der Ausgabe widerspiegelt.

Listing 12.14 ASP zur Anzeige der Werte eines FORM aus einem POST-Request

```
<HTML><BODY>
<%
response.write("<H3>Form Content</H3>")
for each formfield in Request.Form
response.write(formfield&"="&Request.Form(form-
field)&" ")
next
%>
</BODY></HTML>
```

12.2.7 ASP Application Variables

ASP erlaubt es, globale Variablen zu definieren, die entweder pro Session oder systemweit gültig sind. Damit können Sie Werte über mehrere Aufrufe hinweg persistent halten. Globale Variablen können nur einfache Typen sein, keine Verweise auf COM-Objekte.

Listing 12.15 Beispiel zum Setzen einer globalen Variablen **Zähleranwendung**

```
<html><body>
<H3>Here is the value of the counter.</H3>
<P>
<% Application.Lock %>
<% Counter = Application("AccessCounter") %>
<% Counter = Counter + 1 %>
<% Application("AccessCounter") = Counter %>
<% Application.Unlock %>
<% = "<P>The value of the counter is " %>
<% = Counter %>
<% = "</P>" %>
<% = "<P>Connection String" %>
<% = Application("ConnectionString") %>
<% = "</P>" %>
<%  %>
<% Application.UnLock %>
</P></body></html>
```

Das gezeigte Beispiel in Abbildung 12.5 müssen Sie in zwei Browser-Fenstern gleichzeitig aufrufen und beobachten, wie die Zählerwerte sich verändern.

Abbildung 12.4 Beispiel zur Verwendung einer globalen ASP-Variable

12.2.8 Global.asa – Die ASP-Autoexec-Datei

Zur Initialisierung von global gültigen Variablen steht die Datei *GLOBAL.ASA* zur Verfügung. Dies ist ein besonderes ASP-Script, das beim Neustart des IIS automatisch gefunden und ausgeführt wird. Das folgende Listing zeigt ein initiales *GLOBAL.ASA*, das Eventhandler für bestimmte IIS-Systemereignisse setzt. Die Eventhandler werden dann aufgerufen, wenn das entsprechende Ereignis eintritt.

Listing 12.16 Initiale GLOBAL.ASA mit Dummy-Eventhandlern

```
<script language=vbscript runat=server>
SUB Application_OnStart
END SUB

SUB Application_OnEnd
END SUB

SUB Session_OnStart
END SUB

Sub Session_OnEnd
END SUB
</script>
```

Falls in dieser Datei ein Fehler auftritt, erhält der erste Aufrufer einer ASP-Seite folgende Fehlermeldung:

Listing 12.17 Beispiel einer Fehlermeldung in GLOBAL.ASA

```
Script blocks must be one of the allowed Global.asa pro-
cedures. Script directives within <% ... %> are not
allowed within the global.asa file. The procedure names
allowed are Application_OnStart, Application_OnEnd,
Session_OnStart, or Session_OnEnd.
```

Falls Sie *GLOBAL.ASA* geändert haben, müssen Sie den IIS neu booten, um die Änderung wirksam werden zu lassen.

Abbildung 12.5 Neustarten von IIS

12.2.9 Extra-Beispiel: File-Liste mit ASP

Das folgende Beispiel verwendet das `FilesystemObject` von Microsoft, um die Dateien, die sich im aktuellen Verzeichnis auf dem Server befinden, aufzulisten.

Abbildung 12.6 Dateiliste eines Verzeichnisses auf dem Server mit ASP

Listing 12.18 Anzeige der gefunden Verzeichnisse auf dem Server mit ASP

```
<%
Set fs = CreateObject("Scripting.FileSystemObject")
Set subdir = fs.GetFolder(".")
ii = 0
response.write subdir.path
For Each xfile In subdir.Files
```

```
ii = ii + 1
response.write "<P>"
response.write ii
response.write ": " + xfile.Name + "</P>"
Next
%>
```

12.3 Entwicklung von Business Server Pages für den SAP Web AS

Server Pages mit ABAP

Business Server Pages (BSP) sind SAPs Implementierungen von dynamischen Webseiten. BSP sind weitgehend identisch mit Microsofts Active Server Pages und Java Server Pages. BSP werden im Repository des Web AS gespeichert und unterstützen als Programmiersprache für die eingestreuten Scripts ABAP. Der folgende Text wird nur noch die Unterschiede und Ergänzungen zu ASP erwähnen und setzt voraus, dass Sie das im Grundlagenkapitel über ASP Beschriebene bereits kennen.

12.3.1 »Hello World« für BSP

Bearbeiten von BSP mit SE80

BSP werden wie andere Entwicklungen auch mit der Transaktion SE80 im Object Navigator bearbeitet. Zu der vertrauten Umgebung ist eine neue Option **BSP Application** hinzugekommen. (Nebenbei bemerkt: *Entwicklungsklassen* wurden in *Packages* umbenannt).

Das »Hello-World«-Beispiel aus Abbildung 12.11 hat bereits einen Link auf eine weitere BSP mit dem Namen *currencies.htm*, das das nächste Beispiel werden wird.

Abbildung 12.7 BSP-Editor in Transaktion SE80

12.3.2 BAPI-Aufruf aus einer BSP

Ausführung von ABAP Objects-Statements

Innerhalb einer BSP können beliebige ABAP Objects-Statements folgen. Dabei ist diesmal wirklich ABAP Objects gemeint und nicht das umfassendere ABAP IV. Alle Restriktionen der Objektumgebung gelten auch für BSP, zum Beispiel sind keine Kopfzeilen (`WITH HEADER LINE`) für interne Tabellen zulässig. Das nun folgende Beispiel ruft den BAPI-Baustein BAPI_EXCHRATE_GETCURRENTRATES auf, um die aktuellen Währungskurse aus SAP R/3 zu lesen und in HTML aufzulisten. Das Ergebnis sehen Sie in Tabelle 1.2.

Listing 12.19 BSP-Code zum Aufruf von BAPI_EXCHRATE_GETCURRENTRATES

```
<%@page language="abap"%>
<html>
  <head>
    <link rel="stylesheet"
        href="../../sap/public/bc/bsp/styles/
sapbsp.css">
    <title> Hello World BSP Page </title>
<%
DATA: exch_rate_list type bapi1093_0.
DATA: tfrom_curr_range TYPE STANDARD TABLE OF bapi1093_
3.
DATA: tto_currncy_range TYPE STANDARD TABLE OF bapi1093_
4.
DATA: texch_rate_list TYPE STANDARD TABLE OF bapi1093_0.
DATA: treturn TYPE STANDARD TABLE OF bapiret1.

CALL FUNCTION 'BAPI_EXCHRATE_GETCURRENTRATES'
  DESTINATION 'NONE'
  EXPORTING
    date                 = sy-datum
  TABLES
    from_curr_range      = tfrom_curr_range
    to_currncy_range     = tto_currncy_range
    exch_rate_list       = texch_rate_list
    return               = treturn.
%>
  </head>

  <body class="bspBody1">
```

```
    <H1><otr>Listing of exchange rates from
    BAPI_EXCHRATE_GETCURRENTRATES
    through BSP</otr></H1>
     <table border="1" width = "100%">
       <tr>
       <th width="20%">From Currency</td>
       <th width="20%">To Currency</td>
       <th width="60%">Exchange Rate</td>
       </tr>
       <% LOOP AT texch_rate_list into exch_rate_list.
%>
         <tr>
           <td > <%= exch_rate_list-from_curr %> </td>
           <td > <%= exch_rate_list-to_currncy %> </td>
           <td > <%= exch_rate_list-exch_rate %> </td>
         </tr>
      <% ENDLOOP. %>
     </table>
  </body>
</html>
```

From Currency	To Currency	Exchange Rate
AED	JPY	30.55026
AED	USD	0.27241
ARS	BRL	1.20900
ARS	CAD	1.52630
ARS	CLP	471.66000
ARS	COP	1548.00000
ARS	JPY	112.15000
ARS	MXN	9.82810
ARS	PEN	3.16130
ARS	USD	1.00000
ARS	VEB	564.88000
AUD	JPY	69.36541

Tabelle 12.2 Mögliches Ergebnis des Aufrufs von BAPI_EXCHRATE_GETCURRENTRATES

From Currency	To Currency	Exchange Rate
ZAR	JPY	19.06762
ZAR	USD	0.17002

Tabelle 12.2 Mögliches Ergebnis des Aufrufs von BAPI_EXCHRATE_GETCURRENTRATES (Forts.)

Anzeige der BSP-Seite

Auf die neue BSP-Seite kann nun von jedem Webbrowser, der ausreichend Zugriffsrechte auf den Web AS hat, zugegriffen werden. Die korrekte URL wird in den Eigenschaften der BSP-Seite (siehe Abbildung 12.12) angezeigt. Beachten Sie, dass der dort angezeigte Domain-Name gegebenenfalls durch die IP des Web AS ersetzt werden muss, beziehungsweise Sie den Domain-Namen im DNS-Server oder die lokale *hosts*.-Datei eintragen müssen.

Abbildung 12.8 Eigenschaften der BSP-Seite currencies.htm

Folgendes ist eine Beispiel-URI zum Aufruf der BSP-Seite:

```
http://192.168.69.111:8080/sap/bc/bsp/sap/ZAXX_HELLO_
WORLD/currencies.htm
```

12.3.3 Online Text Repository

BSP unterstützt automatische Fremdsprachen

Ein besonderes Feature bietet BSP aber doch in Bezug im Vergleich zu ASP oder JSP. BSP ist in der Lage, alle Texte, die potenziell übersetzt werden müssen, in einem *Online Text Repository* mitzuführen. Die Texte darin können dann mit den bekannten R/3-Übersetzungstools in eine andere Sprache gebracht werden. Beim Aufruf der BSP aus einer anderen Sprachumgebung heraus werden dann automatisch die richtigen Übersetzungen eingefügt.

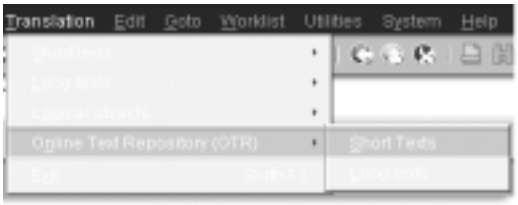

Abbildung 12.9 Übersetzung von OTR-Texten mit SE63

Zum Schluss dieses Kapitels erhalten Sie noch eine Übersicht, die die wichtigsten Direktiven für eine BSP zeigt.

Direktive	Bedeutung
`<%@ page language=("ABAP") %>`	Legt fest, dass die Script-Befehle auf der Seite in ABAP codiert sind.
`<%-- auskommentierter Code oder Text --%>`	Kommentarblock: Der zwischen den Klammern angegebene Text wird weder in die Ausgabe-HTML-Seite übernommen noch als Programmcode interpretiert.
`<% inline code %>`	Alle ausführbaren Statements müssen zwischen `<%` und `%>` stehen.
`<%@ include file="URL"%>`	Der Inhalt der angegebenen Datei wird an der Stelle eingefügt.
`<otr>` beliebiger HTML-Text, kann auch Skripting-Code enthalten `</otr>`	Ein Text zwischen `<otr>` `</otr>` wird automatisch in das Online Text Repository (OTR) eingefügt. Die Texte im OTR können mit Transaktion SE63 übersetzt werden. Wird die BSP dann mit einer anderen Sprachanmeldung aufgerufen, wird der übersetzte Text an Stelle des ursprünglich erfassten Textes ausgegeben.

Tabelle 12.3 Direktiven in BSP