Programmierung mit dem Business Server Page Framework
SAP Netweaver Application Server Abap

Prof. Dr. H. Neuendorf

herbert.neuendorf@mosbach.dhbw.de

16 h Vorlesung +

Testat in Gruppen

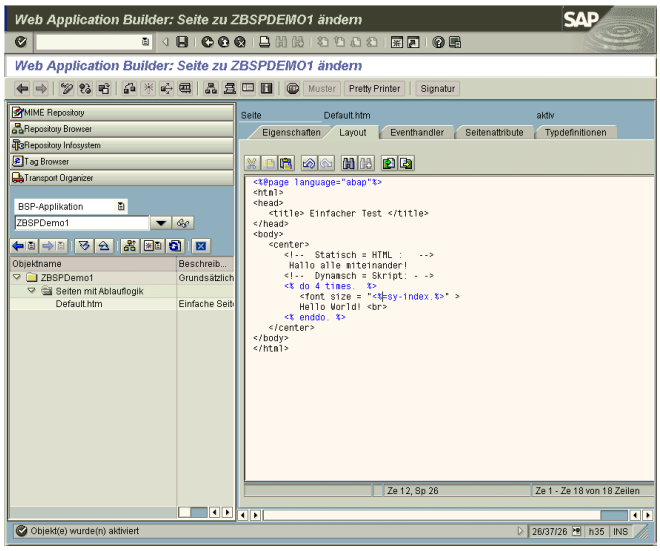
Skript :

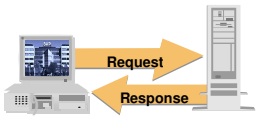
Neuendorf →

Aktuelle Lehrveranstaltungen

BSP_1.pdf

BSP_2.pdf





DHBW Mosbach

- **Literaturhinweise :** In aufsteigender Komplexität - alle Titel in Bibliothek vorhanden:

  [WOLF03] **F.Wolf :** "SAP Web Application Server", 2003.

  - Kompakte und leicht lesbare Einführung in die BSP-Programmierung.

  [HEIN04] **F.Heinemann, C.Rau :** "Webentwicklung in ABAP mit dem SAP Web Application Server", 2.Auflage, 2005.

  - Technisch ausführlicheres Werk (u.a.) über BSP-Programmierung.

  [MCKE06] **Brian McKellar, Thomas Jung:** "Advanced BSP Programming", 2006.

  - Anspruchsvolle Darstellung fortgeschrittener BSP-Programmiertechniken.

- Das vorliegende Skript soll eine Überblick über die architektonischen Prinzipien und Grundlagen der **BSP-Programmierung** im Rahmen des **Internet Communication Frameworks (ICF)** geben.

Business Server Pages BSP

Entwicklung SAP NW : ICF = Internet Communication Framework

Integration :

Komplett in **Workbench** (SE80) integriert ⇒ Debugging, Prüfungen, Transportwesen ...

Komplett in **AppServer** integriert ⇒ DB-Zugriffe + Aufruf von ABAP-Funktionen …

Komplette **ABAP**-Integration ⇒ Alle ABAP-Sprachmittel zVfg.

High-Level Verwendung **ICF** mittels **Server Pages** = **HTML + serverseitiges Skripting**

Weitere Bestandteile :

MIME Repository

→ Ablage MIME-Objekte

Service-Verwaltung des ICF

→ Hinterlegen Verwaltungsinformationen / Konfiguration

…

- Es ist auf ICF-Basis möglich, **Server Pages** zu schreiben - völlig analog zu Java Server Pages, ASP, PHP. Die SAP-Variante wird als **Business Server Pages** bezeichnet (BSP).
- Eine BSP-Seite wird über eine URL im Browser gestartet bzw. auf dem Netweaver Application Server ausgeführt und die generierte Antwort im Browser angezeigt. Eine BSP-Applikation ist im Applikationsserver vollständig integriert. Es können somit innerhalb einer HTML-Seite z.B. Inhalte einer SAP-DB-Tabelle angezeigt werden, die zuvor aus der DB gelesen wurde.

Business Server Pages BSP

High-Level Verwendung **ICF** mittels **Server Pages** = HTML + **serverseitiges** Skripting

HTML = **statischer** Anteil

(**Layout, Präsentation**)

Serverskript = **dynamische**r Anteil

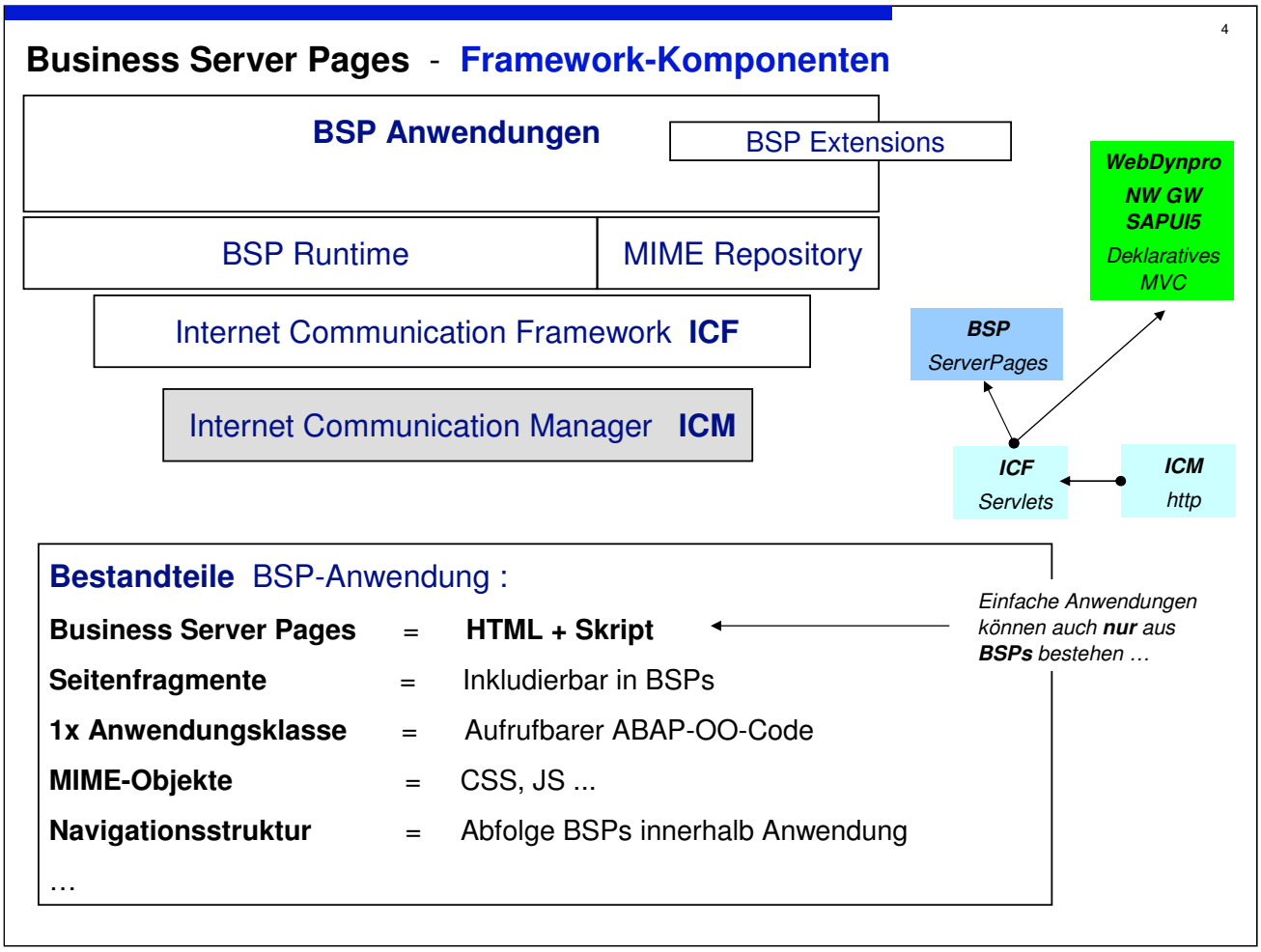
(**Business Logik**, Datenbeschaffung, Eingabeauswertung …)

Serverside-Skriptsprache : **ABAP** (oder JavaScript)

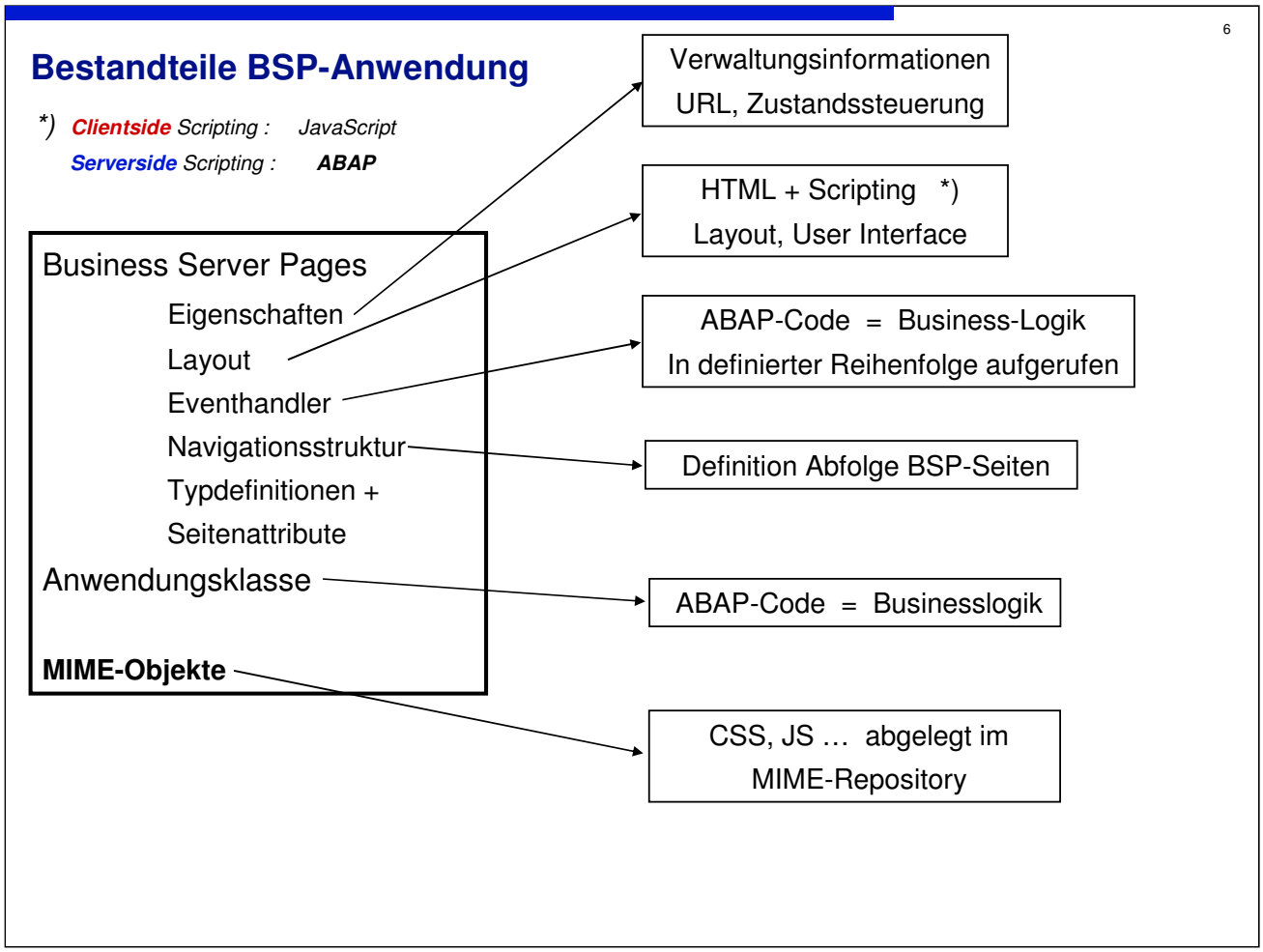
Sprachfestlegung : **<%@ page language = "abap" %>** ← *Keine Spaces zwischen % und @ !!*

Inline Coding : **<% … %>**

- Als serverseitige Skriptsprache können ABAP oder JavaScript im Inline Coding der BSP-Seiten verwendet werden.
- Durch die Seitendirektive **<%@page language = "..." %>** wird die serverseitige Skriptsprache des Layouts festgelegt.
- Um zu testen, ob der eigene **SAP-NWAS via HTTP ansprechbar** ist, dient der Report **RSHTTP01**. Dieser ruft einen ICF-Service des WAS und gibt die komplette HTTP-Antwort aus, die vom Server zurückgesendet wurde. Weitere Reports zum Testen der Internet-Verbindung zu einem Ziel-Rechner sind **RSHTML01** und **RSDEMO_HTML_VIEWER**.



- **Anwendungsklasse**: Man kann einer BSP-Anwendung eine Anwendungsklasse zuordnen. Diese ist eine globale ABAP-Klasse, deren Methoden während des Prozessierens der BSP-Seite aufgerufen werden können. Innerhalb der Anwendungsklasse lässt sich die Business-Logik der BSP-Anwendung kapseln.
- **MIME-Objekte** sind Grafiken, Style-Sheets, Audio- und Video-Dateien; diese werden im MIME-Repository innerhalb der SAP-NW-DB abgelegt zur Einbindung in BSP-Seiten.
- Ein **Thema** ist ein Container für binäre MIME-Objekte. Es wird verwendet, um das Layout (Erscheinungsbild) von Seiten zu ändern, ohne den Layout-Quelltext anpassen zu müssen.
- Unter einer **Navigationsstruktur** versteht man die mögliche Abfolge der BSPs innerhalb einer Anwendung.



- Eine **BSP-Anwendung** setzt sich zusammen aus den **Komponenten**:

  Business Server Pages: BSPs sind die verschiedenen Webseiten, die bei Aufruf im Browser angezeigt werden. Sie enthalten statischen HTML-Code und dynamischen Script Code (ABAP oder Javascript). Dieser Script Code wird auf dem Server ausgewertet. Serverseitige Skriptsprachen dürfen ***nicht*** im Layout einer BSP *gemischt* werden.

  Navigationsstruktur: In der Navigationsstruktur kann festgelegt werden, von welcher Seite zu welcher Folgeseite navigiert werden soll.

  Anwendungsklasse: Die Business-Logik einer BSP-Anwendung kann in einer Anwendungsklasse gekapselt werden, die den Zugriff auf Business-Daten implementiert (z.B. DB-Abfragen). Jede Seite einer BSP-Anwendung kann Attribute und Methoden dieser Klasse ansprechen und verwenden.

  MIME-Objekte: Im SAP-System werden alle MIMEs wie Grafiken, Style Sheets, Java Script, Audio- und Videodateien usw. im zentralen MIME-Repository verwaltet. Mit jeder neu angelegten BSP-Anwendung wird automatisch ein gleichnamiges Verzeichnis im MIME-Repository angelegt. Das MIME-Repository dient zur Ablage von MIME-Objekten, die für die BSP-Anwendung gebraucht werden. Die MIME-Objekte werden vom lokalen Rechner in das MIME-Repository (Teil der SAP-DB) importiert.

  Thema: Ein Thema bildet einen Container für MIME-Objekte, die verwendet werden können, um das Erscheinungsbild einer BSP-Anwendung nachträglich zu verändern. Jedes MIME-Objekt der Anwendung kann durch ein anderes Objekt ersetzt werden - ohne dass der HTML-(Layout-) Code oder die Ablauflogik verändert werden müssten.

Business Server Pages BSP

```
<%@page language = "abap" %>
<!doctype html>
<html>
<head>
  <title> Simple example </title>
</head>
<body>
  <center>
    <!-- Statisch = HTML : -->
     Hallo alle miteinander!
    <!-- Dynamisch = Skript: -->
    <% do 4 times. %>
      <font size = <%=sy-index %> >
      Hello World! <br>
    <% enddo. %>
  </center>
</body>
</html>
```

Aufbau BSP :

1. Eigenschaften :

Komplette Seite (mit URL) - oder Seitenfragment

2. Layout :

Seitengestaltung = HTML + Skript

3. Seitenattribute :

Schnittstelle der BSP-Seite =

Variablen, Referenzen, interne Tabellen …

4. Typdefinitionen :

Definition von ABAP-Typen, auf die innerhalb Seite zugegriffen werden kann.

5. Eventhandler :

Werden beim Verarbeiten einer BSP in definierter Reihenfolge durchlaufen =

Zeitlicher Ablauf der Seiten-Prozessierung

- Im Rahmen der Gestaltung einer BSP-Anwendung muss man sich mit den **Bestandteilen einer BSP** auseinandersetzen - insbesondere den Eventhandlern.
- Während es theoretisch möglich ist, sämtliche Teile der Anwendung in ein oder mehrere reine BSP-Seiten zu stecken, sollte man (wenn immer möglich) das **Layout von der Businesslogik trennen**. Dazu dienen die **Eventhandler mit ABAP-Code** , zur Beschaffung und Manipulation von Daten im Rahmen des abgebildeten Geschäftsvorgangs (Businesslogik).
- Die BSP-Anwendung erzeugt auf der Basis der enthaltenen Programmierbefehle dynamisch HTML, das mit Anwendungsdaten angereichert ist und als reiner HTML-Strom direkt über das HTTP(S)-Protokoll an den Webbrowser des Anwenders geschickt wird.
- Der NWAS verfügt auch über **serverseitiges Scripting** in **JavaScript**. Hierzu wurde ein **JavaScript-Prozessor** in den Kernel integriert, der über die Methoden der Klasse *CL_JAVA_SCRIPT* aus BSP-Anwendungen angesprochen werden kann.

Anlegen BSP-Anwendung

1. **Object Navigator** (SE80) starten + **Repository Browser** anwählen
2. **Paket** wählen & Paketnamen ***Zxxxxxxx*** eingeben
3. Kontextmenü Paket : ***Anlegen*** → *BSP-Bibliothek* → ***BSP Applikation***

   ⇒ System führt durch Schritte zum Anlegen einer neuen BSP-Anwendung

   Abfrage von : Kurzbeschreibung, Paketzuordnung …
4. Kontextmenü BSP-Anwendung : ***Anlegen*** → ***Seite***

   *Seite mit **Ablauflogik*** wählen !

   ⇒ Im **Popup** Abfrage von Seitenpfad + Kurzbeschreibung

   ⇒ Seite wird angelegt
5. **Eigenschaften und Layout** der *Seite* bearbeiten:

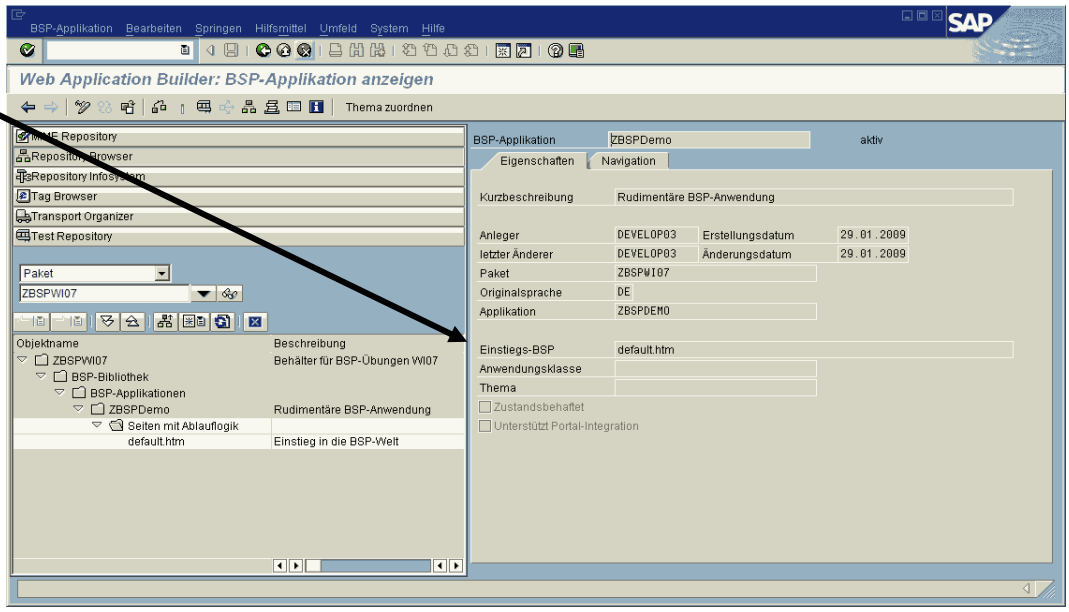
   Generiertes Coding löschen - nur ABAP-Seitendirektive stehen lassen

   HTML + ABAP eingeben & Syntaxprüfung durchführen
6. Compilation durch Kontextmenü Anwendung : ***Aktivieren***
7. Aufruf Anwendung aus SE80 durch Kontextmenü : ***Testen***

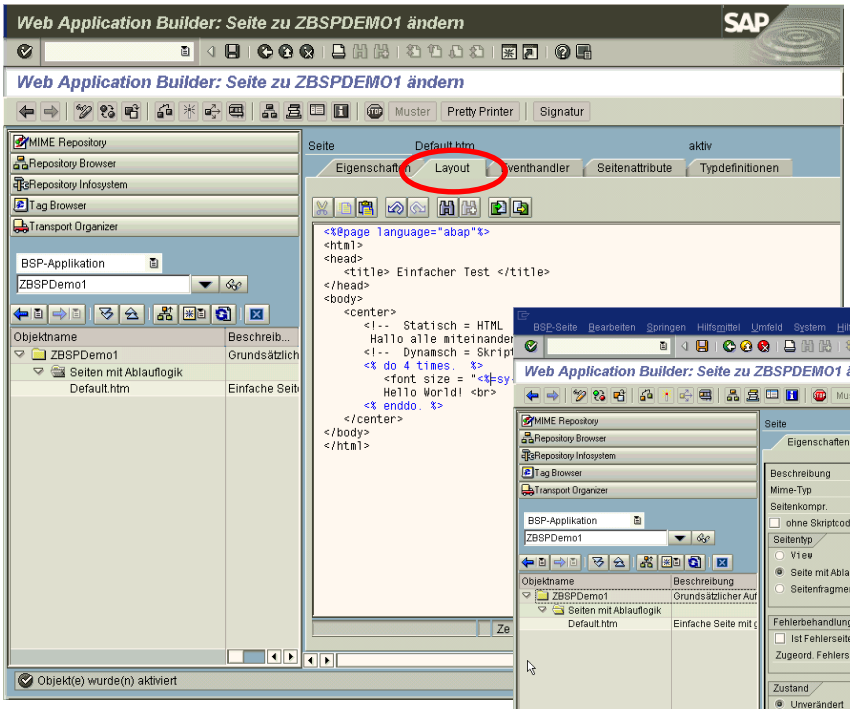
Zugehöriger ***Service*** muss **aktiv** sein ⇒ TA **SICF** **!**

- Jeder BSP-Anwendung **kann** eine globale ABAP-**Anwendungsklasse** zugeordnet werden, die im ClassBuilder (SE24) gepflegt wird. Deren Methoden können in allen BSP-Seiten verwendet werden und sollten wiederverwendbare Businesslogik kapseln (s.u.).
- In den **Eigenschaften** der BSP-Applikation kann die ***Einstiegsseite*** festgelegt werden: Falls die BSP-Anwendung aus mehreren Seiten besteht, kann eine eindeutige Startseite angegeben werden. Diese wird automatisch aktiv, wenn die Anwendung aufgerufen wird.

Anlegen von Ordnern:
Wenn man für **Namen** der **BSP-Seiten** beim Anlegen Pfad angibt, dann wird die entsprechende Ordner-Struktur mit Unterknoten angelegt. So lässt sich die Anwendung strukturieren. zB :
public/ start.htm
public/ flights.htm
private/ customer.htm

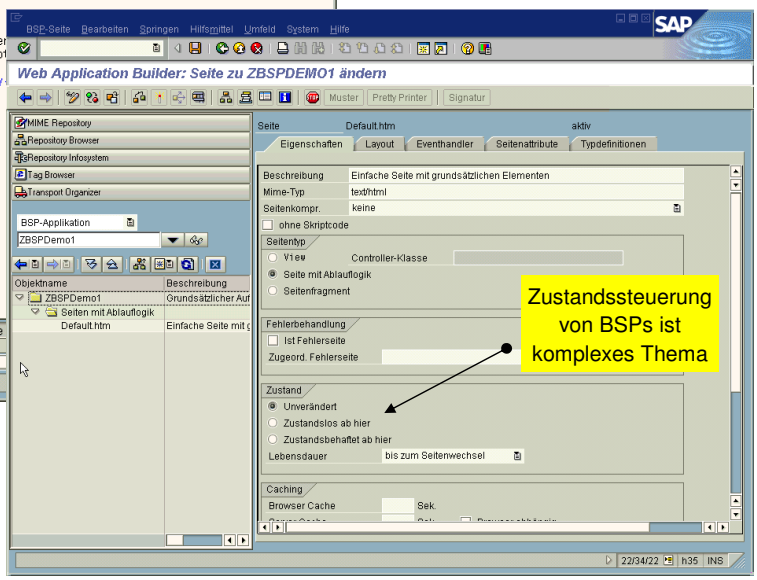


Anlegen BSP-Anwendung



Layout :

HTML + Scripting

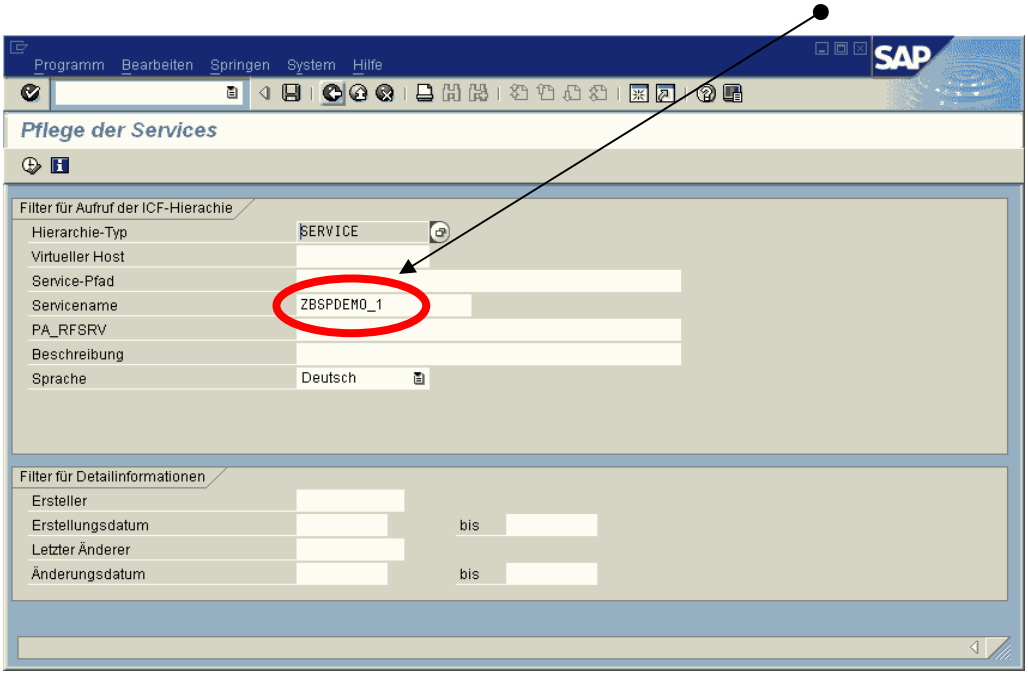


Eigenschaften Seite :

Beschreibung, Seitentyp, Zustandsverhalten, **URL** …

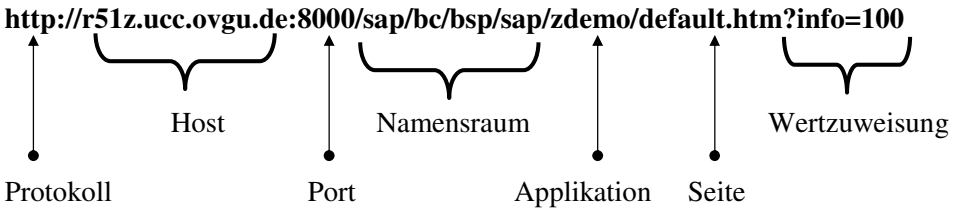
Seitenfragmente haben ***keine*** URL

- Die URLs der einzelnen Seiten können der **Eigenschaftsseite** der Seite entnommen werden.
- Die **Lebensdauer** bestimmt, wie lange die Seite im **Cache des ICM** gehalten wird. Da die meisten Seiten dynamische Inhalte aus der DB zeigen, ist sinnvollerweise "Bis zum Seitenwechsel" voreingestellt - längeres Caching ("session") ist nur bei statischen Seiten sinnvoll.
- Servicepflege in Transaktion: **SICF** *(**Servicename** vorgeben !!)*



URL BSP-Anwendung ↔ Service im SICF

Protokoll :// Host : Port / Namensraum / Applikationsname / Seite ? Parameter



Es genügt URL bis zum **Applikationsnamen** – denn **Startseite** ist in Eigenschaften der BSP-Anwendung definierbar.

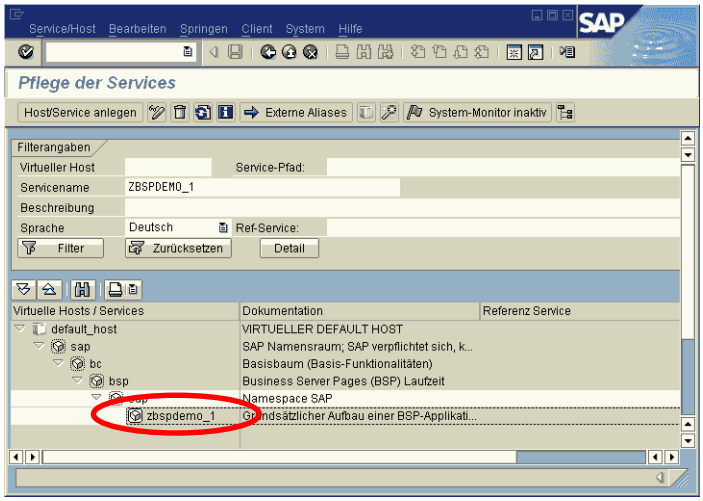
TA **SICF**

Doppelklick auf Service-Eintrag :

Pflege des Service

Kontextmenü :

Service Aktivieren / Deaktivieren



- **Der Aufbau der URL** entspricht dem Standard:

  An erster Stelle wird das *Übertragungsprotokoll* angegeben: Nur **http** oder **https**.

  Anschließend wird der *Host* genannt, indem IP-Adresse oder DNS-Name angegeben wird (University Competence Center Otto-von-Guericke-Universität Magdeburg: ucc.ovgu). Die *Port-Nummer* nennt den Ausführungsport der Applikation.

  Im Anschluss erfolgt die Angabe des *Namensraums* der Applikation im SAP-System. Es schließt sich an der *Name der BSP-Applikation* und (optional – denn die Startseite ist ja hinterlegt) der Name der BSP-HTML-Seite. **Wichtig: In einem Namensraum** müssen die **BSP-Anwendungsnamen eindeutig** sein – auch wenn sie zu verschiedenen Paketen gehören! Gleiche Namen sind nur in verschiedenen Service-Namenräumen (s.u.) möglich !!

  Nach Angabe der BSP-Seite ist es *möglich*, *Parameter* an die Applikation zu übergeben. Diese werden vom Pfad mit "?" abgetrennt. Man könnte einer Variable der Seite einen Wert zuweisen, indem nach dem "?" die Variable mit einem konkreten Wert gleichgesetzt wird.

- Die URL, die nach Aufruf einer Seite im Browser angezeigt wird, ist nicht dieselbe, die für eine Seite in der SE80 angezeigt wird, da von der BSP-Laufzeit zusätzliche Angaben in die URL hineincodiert werden (**URL-Mangling**). Den codierten Inhalt kann man sich durch die SAP-BSP-Anwendung **decode_url/default.htm** anzeigen lassen.
- **Service-Definition:** Die Applikation kann über ihre URL via Intra- oder Internet nur erreicht werden, wenn für sie ein **Service** *angelegt und aktiviert* ist. Die Services sind hierachisch gegliedert. Die *Servicepflege* erfolgt durch die **Transaktion SICF**; hier kann der **BSP-Service** aktiviert bzw. deaktiviert werden. Auch weitere *spezielle Einstellungen* sind hier möglich, z.B. Hinterlegen von Zugangsdaten, Angabe einer speziellen Fehlerseite, etc.
- Für jede neue BSP-Applikation wird ***automatisch*** ein eigener aktivierter Service angelegt. Man kann eine komplette BSP-Applikation einfach **durch *Deaktivieren* ihres Service aus dem Netz nehmen**, um schnell auf Verarbeitungsfehler oder Sicherheitsprobleme zu reagieren.

Servicepflege : SICF

Defaultuser kann hinterlegt werden, um ***anonymen*** Zugriff zu ermöglichen.

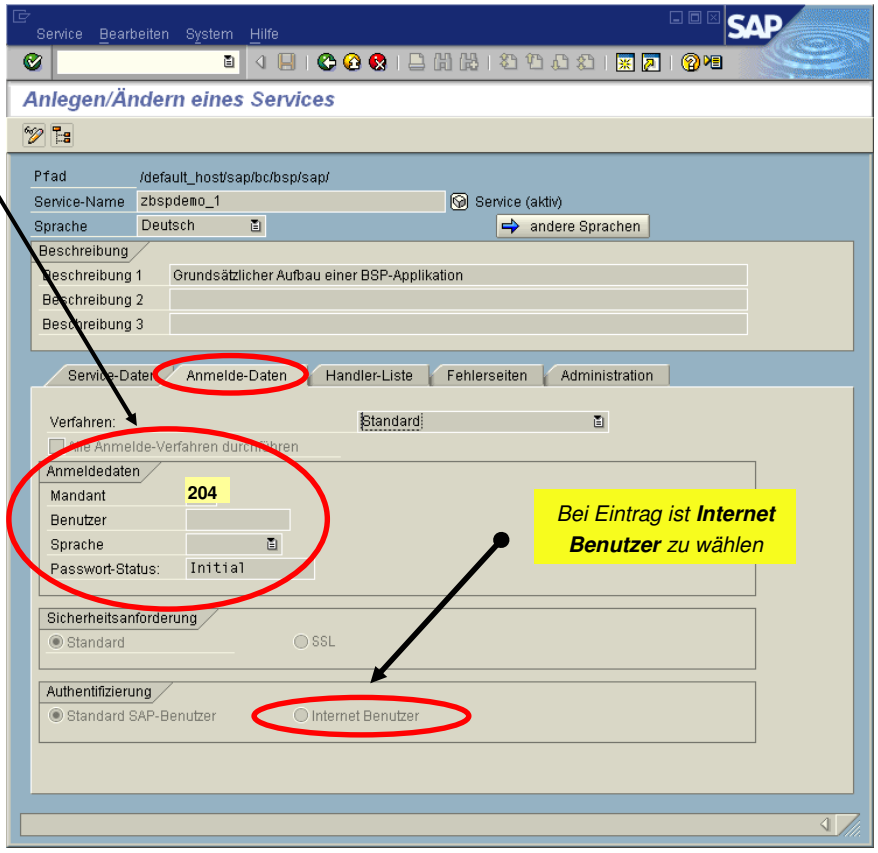
User : **gast**

PW : **mosbach**

XSRF-Schutz deaktivieren !

Sprache aus System-einstellungen ermittelt.

Damit **BSP-Aufruf** durch direkte URL-Eingabe im Browser funktioniert, muss ***Mandant*** hinterlegt sein.



- Die **Zugriffs-Eigenschaften** einer BSP-Anwendung lassen sich in den zugehörigen Service-Eigenschaften pflegen. Hier wird u.a. das Anmeldeverhalten der Anwendung gesteuert.
- Man kann in der **TA SU01** (Benutzerpflege) **Default-Benutzer** (*Service-Benutzer*) mit "anonymen" Zugang auf öffentliche Dienste und minimalen Rechten anlegen. In der **SU01** ist (unter *Logondaten*) Benutzername und Kennwort zu vergeben. Als *Benutzertyp* ist ***Service*** zu wählen, denn nur Servicebenutzer müssen ihr Kennwort nicht ändern und können daher als Default-Benutzer verwendet werden.
- **Arbeitsweise des ICMs:** Bei Ankunft eines Requests sucht der ICM im Servivebaum nach demjenigen Service, der der Request-URL entspricht. Dessen Einstellungen gelten dann für die Request-Bearbeitung
- **Wichtig:** Damit der **Aufruf einer BSP** durch ***direkte Eingabe der URL der Eigenschaftsseite im Browser*** funktioniert, muss in der Service-Verwaltung der ***Mandant*** hinterlegt sein!
- Man erhält für alle unterstützen Protokolle **Informationen** zu Servernamen, Port und Timeout-Einstellungen, wenn man auf der SICF-Servicebaum-Anzeigeseite die **Info-Ikone** wählt.

- **Nachtrag zum Anlegen von BSP-Anwendungen - Anlegen von Ordnern:** Wenn man für die *Namen der BSP-Seiten* beim Anlegen *einen Pfad angibt* statt eines einfachen Namens, dann wird innerhalb der BSP-Anwendung die *entsprechende Ordner-Struktur angelegt*, d.h. es entstehen **Unterknoten**, in denen die jeweiligen Seiten aufgelistet sind. Dadurch lässt sich die Anwendung übersichtlicher strukturieren. Z.B in :

  public/ start.htm

  private/ customer.htm

BSP-Scripting im Layout

- **Clientseitige Skripte JS :** `<script> … </script>`
- **Serverseitige Skripte JS :** `<%@page language="javascript"%>`
- **Serverseitige Skripte ABAP :** `<%@page language="abap"%>`

  In Eventhandlern **nur** ABAP !

Zuweisung von Variablenwerten durch Tag
<%= Variablenname %>

Keine Spaces zwischen % und =

```
<%@page language = "abap" %>
<html>
<head>
  <title> Serverside ABAP </title>
</head>
<body>
 <%-- BSP-Kommentar --%>
  <center>
   <% do 4 times. %>
     <font size = <%=sy-index %> >
     Hello World! <br>
   <% enddo. %>
  </center>
</body>
</html>
```

```
<%@page language = "javascript" %>
<html>
<head>
  <title> Serverside JS </title>
</head>
<body>
 <%-- BSP-Kommentar --%>
  <center>
   <% for( i=0; i<= 5; i++ ) { %>
     <font size = <%=i %> >
     Hello World! <br>
   <% } %>
  </center>
</body>
</html>
```

Serverseitiger Scriptcode

<% ... %>

- Fundamentaler **Unterschied** zwischen **client- und serverseitigem Scripting**:

  Client-Skripte laufen direkt *im Browser* des Anwenders ohne weiteren Netzzugriff. Können nicht auf Dateien oder DB zugreifen, sondern nur mit Elementen der umgebenden HTML-Seite operieren und sind von den Möglichkeiten des ausführenden Browsers abhängig. Clientseitige JS-Skripte werden durch das *Standard-HTML5-Tag* **<script> </script>** eingeschlossen.

  Server-Skripte laufen auf dem Webserver, erzeugen dynamisch HTML-Code und können auf Dateien und DB des Webservers zugreifen. Beim Aufbereiten einer HTML-Seite führt der Server die eingebetteten Skripte aus. Bekannte Serverskriptsprachen sind JSP, ASP, PHP. Der NWAS sieht in BSPs die *SAP-spezifischen* Seitendirektive **<%@page language="..." %>** am Anfang der BSP-Seite vor, um die verwendete serverseitige Skriptsprache festzulegen. Das eigentliche Scriptcoding wird jeweils in das *BSP-Tag* **<% ... %>** eingeschlossen, um es vom umgebenden HTML-Code des Layouts zu trennen.
- Da in den Eventhandlern nur in **ABAP** programmiert werden kann, wird man in der Regel auch im Layout-Teil einer BSP mit **ABAP** entwickeln. Es gelten die Syntaxregeln für ABAP Objects.
- Generell: **Ziel** des server- und des clientseitigen Scriptings ist stets die **Gestaltung des Layouts**. Dagegen sollte die Anwendungslogik einer BSP-Anwendung in die Anwendungsklasse und die Eventhandler ausgelagert werden.
- **Zuweisung von Variablenwerten:** Durch das Tag **<%= *Variablenname* %>** wird der Inhalt der Scriptvariablen an der entsprechenden Stelle in den HTML-Code eingefügt. Es dürfen *keine Spaces* zwischen den Zeichen **%** und **=** stehen.

BSP-Seitenfragmente

*Inkludieren durch Angabe der **relativen URL** der Include-Datei*

about.htm

```
<%@page language="abap"%>
<!--  Einbinden Include-Datei  -->
<%@ include file="navigation.htm" %>

<h3> Portal </h3> <br>
<h4> Formlos - Fristlos - Zwecklos </h4> <br>

</body>
</html>
```

navigation.htm

```
<!--  Include-Datei  -->
<!docctype html>
<html>
<head>
  <title> Portal </title>
</head>
<body>
<h2> Willkommen </h2>
<a href="default.htm"> Einstieg </a>
 <br>
<a href="news.htm"> Aktuelles </a>
<br>
<a href="about.htm"> Kontakt </a>
<hr>
```

Anlegen Seitenfragmente durch :

Anlegen → Seite

und Auswahl von ***Seitenfragment*** im Popup.

Generiertes HTML im Browser :

Include-Code wird einfach eingefügt.

- **Seitenfragmente** sind eine **Modularisierungstechnik**. Eine BSP, die *beim Anlegen* und in ihren Eigenschaften *als Seitenfragment* gekennzeichnet ist, ist *nicht ausführbar* und besitzt keine eigene URL, keine eigenen Seitenattribute, Typdefinitionen und Eventhandler. Vielmehr kann sie die Typdefinitionen, Seitenattribute und Data-Anweisungen der inkludierenden BSP-Seite "mitbenutzen".
- **Nicht ausführbar** heisst: Sie kann nicht über eine eigene URL angesprochen werden oder als Link in einer anderen Seite benutzt werden.
- Seitenfragmente enthalten HTML-Code und evtl auch MIME-Objekte. Sie werden durch den WAS in die Hauptseite **eingebunden**, dh stellen eine serverkontrollierte Seitenaufteilung dar. Ein Seitenfragment wird über die BSP-Include-Direktive eingebunden.
- Technisch sind *bedingte Einbindungen* möglich – diese sind jedoch meist nicht sinnvoll und führen in der Regel zu nicht-wohlgeformten HTML-Strukturen:

  <% data: n type i. n=10. **if n>100.** %> <%@ include file="nav.htm" %> <% **endif.** %>
- Seitenfragmente sind **schachtelbar**: In ein Seitenfragment kann ein anderes Seitenfragment inkludiert werden.
- Durch Seitenfragmente kann Layout-Coding wiederverwendet werden. Typische Anwendungen sind Logo-Bereiche oder Navigationsleisten. (Falls ein inkludiertes Seitenfragment (noch) nicht vorhanden ist, lassen sich die inkludierenden BSPs nicht kompilieren bzw. aktivieren – man erhält einen entsprechenden Syntaxfehler.)
- Der Inhalt der BSP-Seite und der inkludierten Seitenfragmente sollte sich *insgesamt* zu syntaktisch korrektem HTML **ergänzen**. Ist somit z.B. in einem Top-Seitenfragment bereits eine body-Deklaration enthalten, so sollte diese in der BSP-Seite selbst nicht mehr erscheinen.
- Wenn ein Seitenfragment in **mehreren** BSP-Seiten inkludiert wird und auf **Seitenattribute** zugreift, so müssen die entsprechenden Seitenattribute in **allen** inkludierenden BSP-Seiten deklariert werden.

Mime Repository

Mimes (.css, .js, .gif ...) vom Frontend importiert - werden Teil der Anwendung.

Bilder durch Tag **<img src= " … " >** ins Layout eingebunden.

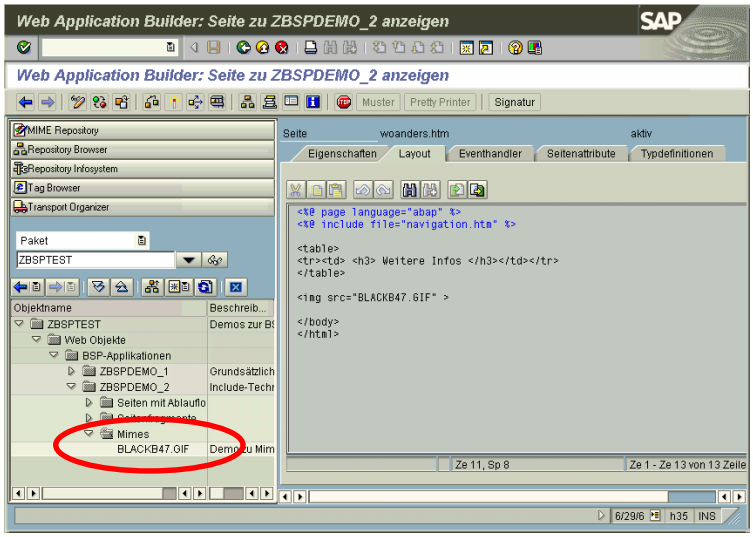
1. Kontextmenü BSP-Anwendung :

   Anlegen → MIME-Objekt → Importieren

2. Am Frontend MIME-Objekt selektieren.

   Wird im **Ordner *Mimes*** als Teil der BSP-Anwendung gelistet.

3. Im *Layout* der BSP-Seite einbinden.



Auf diese Weise können auch **JS- und CSS-Libraries** eingebunden werden.

- Wenn man eine BSP-Anwendung anlegt, wird automatisch im **MIME-Repository** ein gleichnamiger **Knoten** angelegt, der bis zum Löschen der BSP-Anwendung existiert. Hier werden MIME-Objekte der Anwendung abgelegt. Sichtbar durch Drucktaste **MIME Repository** im Object Navigator.
- Durch Rechtsklick (Kontextmenü) auf den angelegten Knoten kann man unterhalb der Knotenebene der BSP-Anwendung noch *zusätzliche Ordner* einfügen und einen **Hierarchiebaum anlegen**. Auch in der Darstellung der BSP-Anwendung im Repository Browser kann über das *Kontextmenü* des MIME-Ordners ein Unterordner angelegt werden.
- MIME-Objekte lassen sich in den WAS **importieren**. Dabei werden die MIMEs in speziellen DB-Tabellen abgelegt. MIME-Objekte können in der SE80 **nicht kopiert** werden; dadurch wird das Erzeugen von Dublikaten *bewusst* erschwert. Da MIME-Objekte sehr groß werden können, sollten sie nur *einmal* in der DB gespeichert werden. Sinnvolle Inhalte sind: .css- oder auch .js-Dateien.
- Mit dem SAP-ABAP-Programm **BSP_UPDATE_MIMEREPOS** können Objekte aus dem MIME Repository bequem exportiert und importiert werden. Insbesondere kann der komplette MIME-Repository-Inhalt von BSP-Anwendungen auf den lokalen Rechner exportiert werden.
- Alle verwendete MIMEs werden **ICM-Server-Cache** ge-chacht - andernfalls würden die DB-Zugriffszeiten kritisch ins Gewicht fallen.

Anpassen Layout - Einsatz CSS

Ziel : Anpassen des Layouts - ohne HTML-Quellcodeänderungen

Einheitliche firmenspezifische **Optik (CI)**

Mittel: Trennung von **Struktur und Stil** ⇒ **Cascading Style Sheet** =

Zusammenfassung Layout-Deklaration in separaten File als Formatvorlage.

Einbinden durch <**link**>-Tag im <head> der Seite :

```
<link  type ="text/css"  rel="stylesheet"  href="mysheets.css" >
        bzw. ab  HTML5  :
<link  rel="stylesheet"  href="mysheets.css" >
```

Bezug auf Elemente der CSS-Definition :

```
< body class="body1" >
```

Weitergehende Layout-Definition : **Themenkonzept**

- Web-Anwendungen sollen ein möglichst **einheitliches Erscheinungsbild** haben, damit der Anwender sich leichter auf den Seiten zurechtfindet und Produkt und Anbieter wiedererkennt. Bei vielen Seiten würden lokale Änderungen immer wieder zu erheblichen Arbeitsaufwand führen. Deshalb ist es sinnvoll, das Layout von der Funktionalität der Webseite zu trennen und die Layoutdefinition in einem **separaten CSS-File** abzulegen. Diese Formatvorlage wirkt sich somit auf alle Seiten einer Website aus und Änderungen können durch Editieren des CSS-Files sofort für die gesamte Site realisiert werden. Die CSS-Datei enthält alle relavanten Style-Anweisungen im Klartext. Von jeder Seite werden diese Anweisungen durch das <LINK>-Tag im <HEAD>-Bereich der HTML-Seite geladen.
- Die Syles sind **kaskadierend**, dh sie können im Style Sheet definiert werden, aber gegebenfalls bei bestimmten Seitenobjekten mit einer neuen Definition übersteuert werden. Zur Syntax von CSS-Files sei auf die Fachliteratur verwiesen.
- **Themenkonzept:** Um das Layout einer *BSP-Anwendung als Ganzes anpassen* zu können, setzt man Themen ein. Dies sind eigenständige Repository-Objekte, die im Repository Browser (SE80) gepflegt werden. Wenn ma ein Thema anlegt, kann man aus dem MIME-Repository *MIME-Objekte in ein Thema aufnehmen*. Ein Thema kann einer oder mehreren BSP-Anwendungen zugeordnet werden. Ohne dass die BSP-Applikation modifiziert werden muss, kann das ursprüngliche Layout auf ein anderes (zB firmenspezifisches) Layout umgestellt werden:

  In der *Eigenschafts-Seite der BSP-Anwendung* kann unter **Thema** ein Thema (und somit die zugehörigen MIMEs) der BSP-Applikation zugeordnet werden.

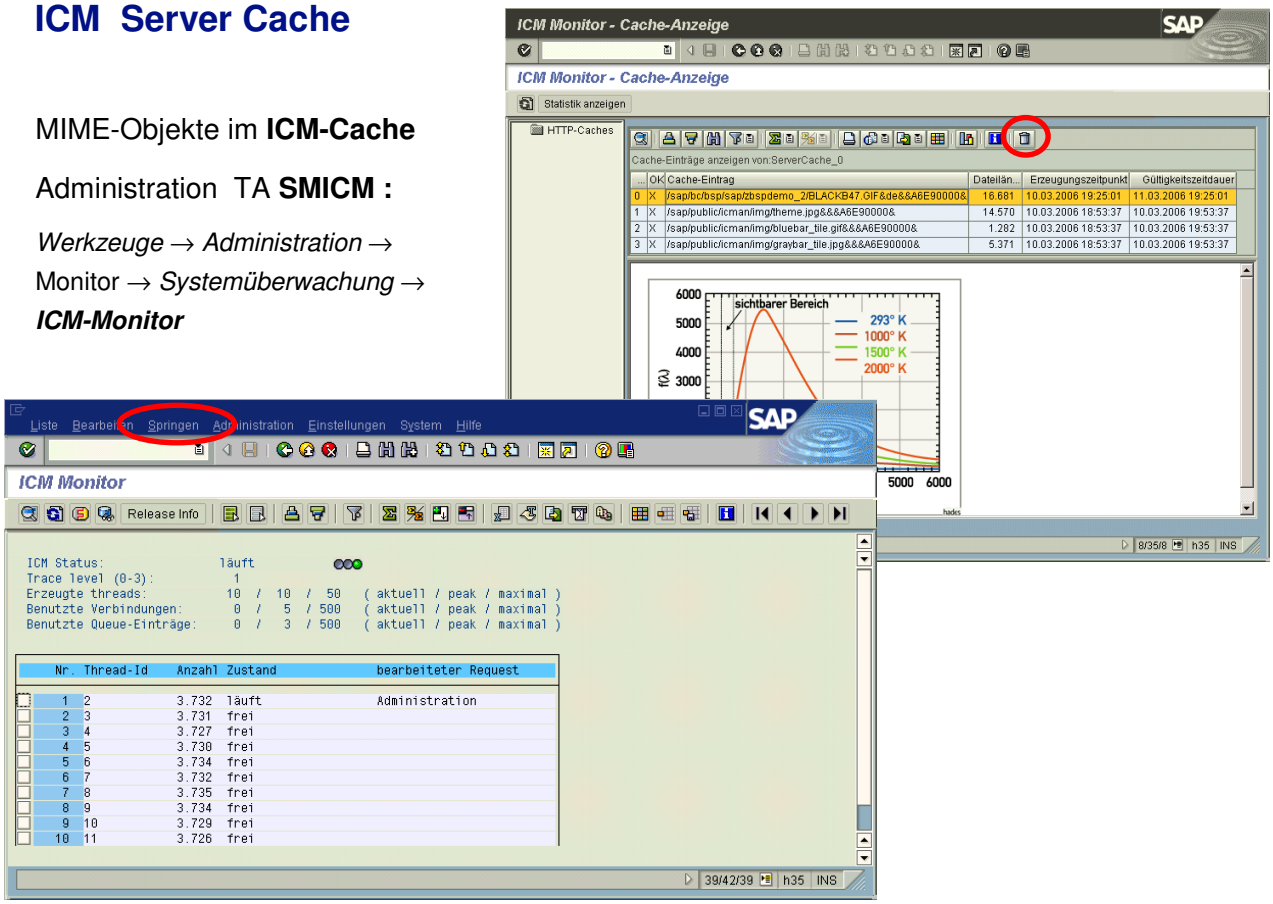
  Auf das Themenkonzept wird hier nicht weiter eingegangen - wir verweisen auf weiterführende Literatur, insbesondere [MCKE06].

ICM Server Cache

MIME-Objekte im **ICM-Cache**

Administration TA **SMICM :**

Werkzeuge → *Administration* → Monitor → *Systemüberwachung* → ***ICM-Monitor***



- **Performance-Aspekte:** Bei MIME-Objekten handelt es sich häufig um Bilder oder multimediale Objekte, die in der Regel relativ gross (mehrere 100kB) sind. Somit sollten diese nicht bei jedem Zugriff von Neuem aus der DB gelesen werden! Deshalb werden MIME-Objekte beim ersten Zugriff im **ICM-Servercache** des WAS abgelegt und beim erneuten Zugriff aus diesem zurückgeliefert - mit Zugriffszeiten im ms-Bereich.
- Die im Cache abgelegten Objekte besitzen eine **Verfallszeit**, die global über Profilparameter des ICM gesteuert werden (Standardwert: 1 Tag). Danach wir das Objekt ungültig und bei Platzbedarf aus dem Speicher verdrängt.
- In der **Transaktion SMICM** kann man unter: *Springen* → *HTTP Plugin* → *Server-Cache* → *Anzeigen* den **Cache-Monitor** aufrufen und manuell MIME-Objekte aus dem Cache entfernen.
- Anmerkung: CSS- und MIME-Objekte werden sowohl im ICM-Cache als auch im Cache des lokalen Browsers gehalten. Bei Veränderungen muss somit:

  a) der **ICM-Cache ungültig** gemacht werden durch Transaktion SMICM, Menüpunkt: *Springen* → *HTTP Plugin* → *Server-Cache* → *Global invalidieren*

  b) und auch ein ***Refresh*** im Browser durchgeführt werden.

Seitenattribute – Globale Variablen BSP-Seite

Lokale Variablen :

Können mit ***DATA**-Anweisung* im Layout & jedem Eventhandler vereinbart werden.

Sind jedoch *nur* im Layout *oder* jeweiligen Eventhandler sichtbar !

Globale Variablen der **gesamten** BSP-Anwendung :

Könnten als Attribute der ***Anwendungsklasse*** definiert werden.

Globale Variablen der *einzelnen* BSP-Seite :

Können als ***Seitenattribute*** der BSP-Seite definiert werden.

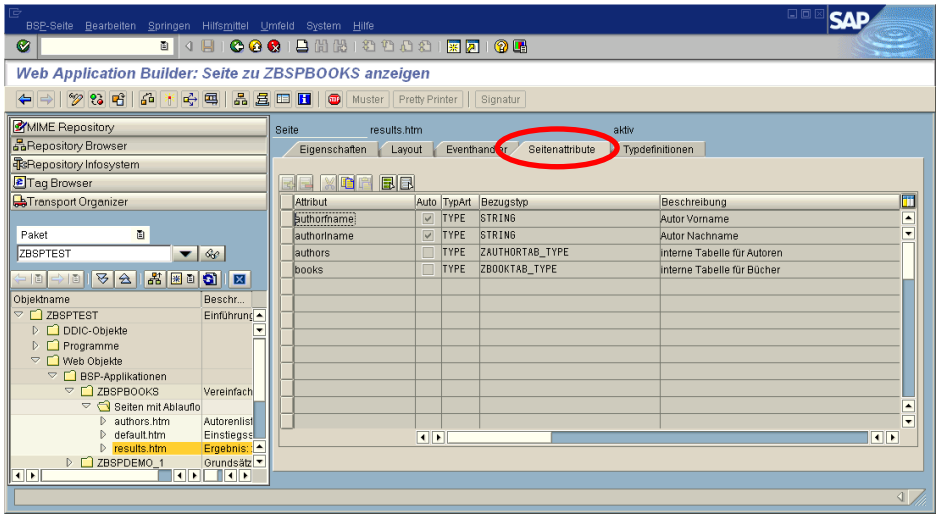
Sind im **Layout** *und* allen **Eventhandlern ihrer** BSP-Seite sichtbar.

Auto-Seitenattribute :

Können beim Aufruf der BSP durch Methode *naviagation→set_parameter( )* gefüllt werden - s.u.

Gültigkeitsbereich nicht identisch mit ***Gültigkeitsdauer*** des Inhalts von Variablen !

Hängt davon ab, ob Anwendung **stateful** oder **stateless** läuft !



- **Seitenattribute** sind Parameter, die für eine BSP explizit deklariert werden. Sie sind zugleich die ***Schnittstelle einer BSP*** nach außen. In Seitenattributen können Felder, Arbeitsbereiche oder interne Tabellen definiert werden - unter Rückgriff auf das ABAP-Dictionary aber auch mittels selbstdefinierter Typen. Die in den Seitenattributen definierten Datenbehälter sind in Bezug auf die **zugehörige** BSP-Seite **global** sichtbar (*Gültigkeitsbereich*): Sie können ohne Neudeklaration im Layout und allen Eventhandlern **ihrer** Seite verwendet werden.
- So kann man typischerweise in Seitenattributen Daten ablegen, die im Eventhandler OnInitialization ermittelt werden, um sie im Layout und anderen Eventhandlern zu nutzen.
- Soll ein Datenobjekt an **mehr als einem Ort** (also neben dem Layout auch in *mehreren* Eventhandlern) zur Vfg stehen, dann ist dieses Datenobjekt als Seitenattribut der BSP zu definieren.

  Muss ein Datenobjekt *nur an einem Ort* zur Vfg. stehen, dann genügt es, das Datenobjekt **lokal** im Layout oder dem entsprechendem Eventhandler mit einer DATA-Anweisung anzulegen.
- **Wichtig**: Der Zugriff auf die Inhalte der Seitenattribute ist nicht zu allen Zeitpunkten gegeben. **Wie lange** eine globale Variable einer BSP-Seite wirklich ihrem zugewiesenen Wert behält (*Gültigkeitsdauer)*, hängt davon ab, ob die Anwendung **stateless oder stateful** abläuft. Dies wird später vertieft.
- **Auto-Seitenattribute**: Die **Übergabe von Datenwerten** von einer BSP-Seite an die Folgeseite wird auch durch Seitenattribute realisiert. Es können einfache Datentypen aber auch ganze interne Tabellen übergeben werden. Beim Aufruf einer BSP-Seite kann die aufrufende BSP-Seite **Werte übergeben**, mit denen die Seitenattribute gefüllt werden. Voraussetzung ist, daß solche Seitenattribute als ***auto*** gekennzeichnet sind (s.u.).

BSP-Typdefinitionen

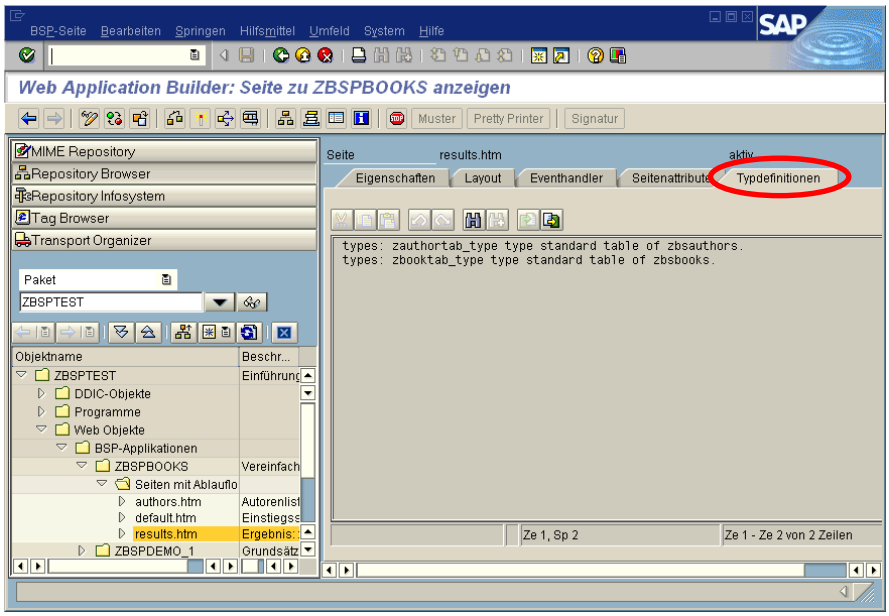
Typdefinitionen in konventioneller **ABAP**-Syntax.

Auf alle ABAP-**Dictionary**-Typen kann referenziert werden.

Definierte Typen stehen zur Vfg. im **Layout**, **Eventhandlern** und auch in **Seitenattributen** der *jeweiligen* BSP-Seite.

Werden *dieselben* Typen in ***mehreren*** Seiten der BSP-Anwendung benötigt,

dann ist *globale* Definition im **Dictionary** sinnvoll.



- Unter dem Reiter **Typdefinitionen** können eigene **ABAP-Typen** definiert werden, auf die ebenfalls *überall* innerhalb einer Seite zugegriffen werden kann – also auch bei der Variablendefinition in den Seitenattributen. Hier kann z.B. der Typ einer internen Tabelle definiert werden, um damit ein Seitenattribut zu typisieren.
- Es besteht in der gesamten BSP-Anwendung Zugriff auf alle global im **ABAP-Dictionary** definierten Typen.

Statische Seitennavigation

Folgeseite wird **fest** vorgegeben.

Einbinden von **Links**, die auf **URL** der Folgeseite verweisen :

```
<a  href="http://www.mosbach.dhbw.de" >   Daheim an der DHBW   </a>
```

Übergabe von Variablen an nächste Seite durch Anfügen von **Name-Value-Paar** an URL :

```
<a  href="http://www.mypage.de/search?var1=wert&var2=wert2" > ... </a>
```

```
<%@page language="abap"%>
<html>
<head>
  <title> Portal </title>
</head>
<body>
  <h2> Willkommen </h2>
  <table>
    <td > <a  href ="default.htm"> Wieder zum Einstieg  </a> </td>
    <td > <a  href ="woanders.htm"> Weitere Infos  </a> </td>
    <td > <a  href ="about.htm"> Ueber uns  </a> </td>
   </table>
</body>
</html>
```

Absolute Links :
Komplette URL

Relative Links :
Relativ zu Ausgangsseite

- Hier wird die **statische** Seitennavigation mittels **Hyperlinks** behandelt, die Standard-Html-Tags verwendet und nicht auf BSP-spezifische Abläufe und Objekte zurückgreift. Diese Art der Implementierung ist einfach, aber statisch. Der Benutzer kann den Navigationsweg nicht durch Eingaben dynamisch gestalten.
- Man unterscheidet **absolute** Links (wie im Text) und relative Links (wie im BSP-Code). Bei **relativen** Links muss nicht die gesamte URL angegeben werden, es reicht der *relative Pfad* zur neuen Seite, ausgehend von der aktuellen Seite und ihrer URL.
- Das Tag **<a href ...> ** kann nicht nur Texte sondern auch Grafiken einschließen. Somit lassen sich Hyperlinks auch auf grafische Elemente der BSP legen.
- Die neue Seite wird vom Browser angefordert. Auch bei der statischen Navigation mittels Hyperlinks werden die **Auto-Seitenattribute der Folgeseite** mit den Werten gefüllt, die durch eine **"?"-Parameterübergabe** in der URL mitgeschickt werden.

Formulare form + action + method post vs. get

Action-Attribut bestimmt aufgerufenes Programm = **ServerPage.**

Formulardaten werden übermittelt.

Füllen **auto-Seitenattribute** der gerufenen BSP mit Werten ... s.u.

```
<%@page language="abap"%>
<html>
<head>  <title> Portal </title>  </head>
<body>
  < form  action="results.htm"  method = "POST"  >
      <input  type = "TEXT"    name = "CUSTOMER" >
      <input  type = "SUBMIT "    name = "user"    value = "Abschicken!"   >
  </ form >
</body>
</html>
```

Flexibler :

Dynamische Seiten-navigation in Event-handlern mittels *navigation*-Objekt …

Attribut **method** bestimmt **Art der Formulardaten-Übermittlung** :

GET Anfrage einer Ressource vom Server – **ohne** dass sich dessen **Zustand** dadurch ändert
Daten des Formulars werden nach **?** in URL in name/value-Paaren (sichtbar) übermittelt.

POST Anfrage mit Datenversand an Server, wodurch sich dessen **Zustand ändert**
Daten werden im HTTP-Body an den Server gesendet.

- Wenn auf der Seite ein **Formular definiert** ist, dann kann mittels **action-Attribut** die Folgeseite *statisch* festgelegt werden. Diese Folgeseite wird vom Server prozessiert und geliefert, wenn der Anwender über eine **SUBMIT-Drucktaste** das Formular zum Server sendet und die neue Seite anfordert. Die im Formular eingegebenen Daten werden als Parameter an die Folgeseite übermittelt und füllen die auto-Seitenattribute der aufgerufenen Seite mit Werten.
- Auf diese Weise wird *pro* Formular *eine* Folgeseite festgelegt. Jedes Formular kann somit nur die Navigation zu *einer* Folgeseite ermöglichen. Nicht jede Anwendung ermöglicht den sinnvollen Einsatz mehrerer Formulare in einer Seite. Somit liefert das **action-Attribut** nur beschränkte Möglichkeiten, die Folgeseiten innerhalb einer Anwendung dynamisch zu steuern.
- **Anmerkung zu Post und Get : Get** wird verwendet, um primär Daten vom Server abzufragen, ohne dass sich dessen Zustand dadurch verändert, d.h. ohne dass z.B. Daten in der Datenbank verändert werden. Auch bei Get können jedoch Daten übermittelt werden – allerdings kann die Datenmenge die an die URL angehängt werden darf (abhängig vom involvierten Server) begrenzt sein. **Post** wird typischerweise verwendet, um Daten an den Server zu senden – danach verändert sich der Zustand des Servers (des Systems) als Reaktion auf die empfangenen Daten – z.B. indem Daten in der zugehörigen Datenbank verändert werden. Die Datenmenge ist nicht begrenzt.
- Ein mit **GET** abgeschicktes Formular kann in die Favoritenliste des Browsers aufgenommen werden. Weil die Formulardaten in die URL hineincodiert werden, kann die URL mit den Formulardaten vom Browser abgespeichert werden. Wird diese URL wieder angefordert, so werden damit die ursprünglichen Formulardaten erneut zum Server geschickt. Dies ist bei **POST** nicht möglich. Da der Browser die angeforderte URL in seiner Adresszeile anzeigt, sieht man bei **GET** Daten, die man eigentlich nicht sehen wollte. Wird aber das Formular über **GET** abgesendet, so tauchen diese in der URL auf und werden deshalb auch in der Adresszeile des Browsers angezeigt. Bei **POST** sieht man diese Daten in der Adresszeile nicht.
- HTTP kennt noch weitere Methoden (Verben) (PUT, HEAD, DELETE, OPTIONS, TRACE, CONNECT), die für uns jedoch hier nicht relevant sind.

BSP-Ereigniskonzept : Trennung Layout / Businesslogik

BSP besitzt **Zeitpunkte**, die über **vordefinierte Eventhandler** abgebildet sind - und in **fest** vorgegebener **Reihenfolge** durchlaufen werden.

Sinn : Koordination / Aufruf der serverseitigen Geschäftslogik

DB-Zugriffe, Aufruf ABAP-Funktionalität.

In Eventhandlern ausschließlich **ABAP**.

- OnCreate
- OnRequest
- **OnInitialization !**
  - *Layout = BSP*
- (OnLayout)
- OnManipulation
- **OnInputProcessing !**
- OnDestroy

Layout :

HTML + Skript

```
HTLM
<% ABAP %>
HTML
<% ABAP %>
HTML
```

ABAP-Eventhandler :

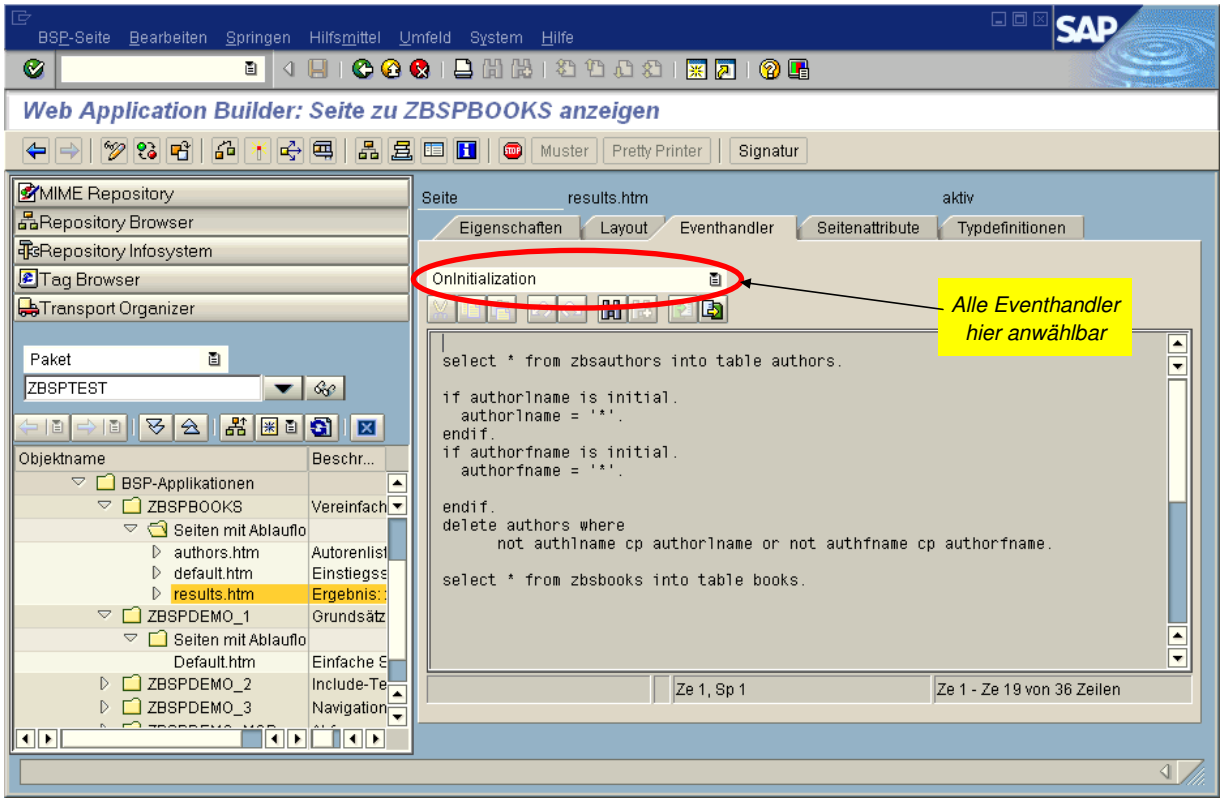
Businesslogik + Datenbeschaffung

```
SELECT * FROM sflight into itab
  …
ENDSELECT.
```

Aufruf der Eventhandler und **Ausführung** ihres ABAP-Codings : Automatisch und in **fester** Reihenfolge durch **BSP-LZ**-Umgebung.

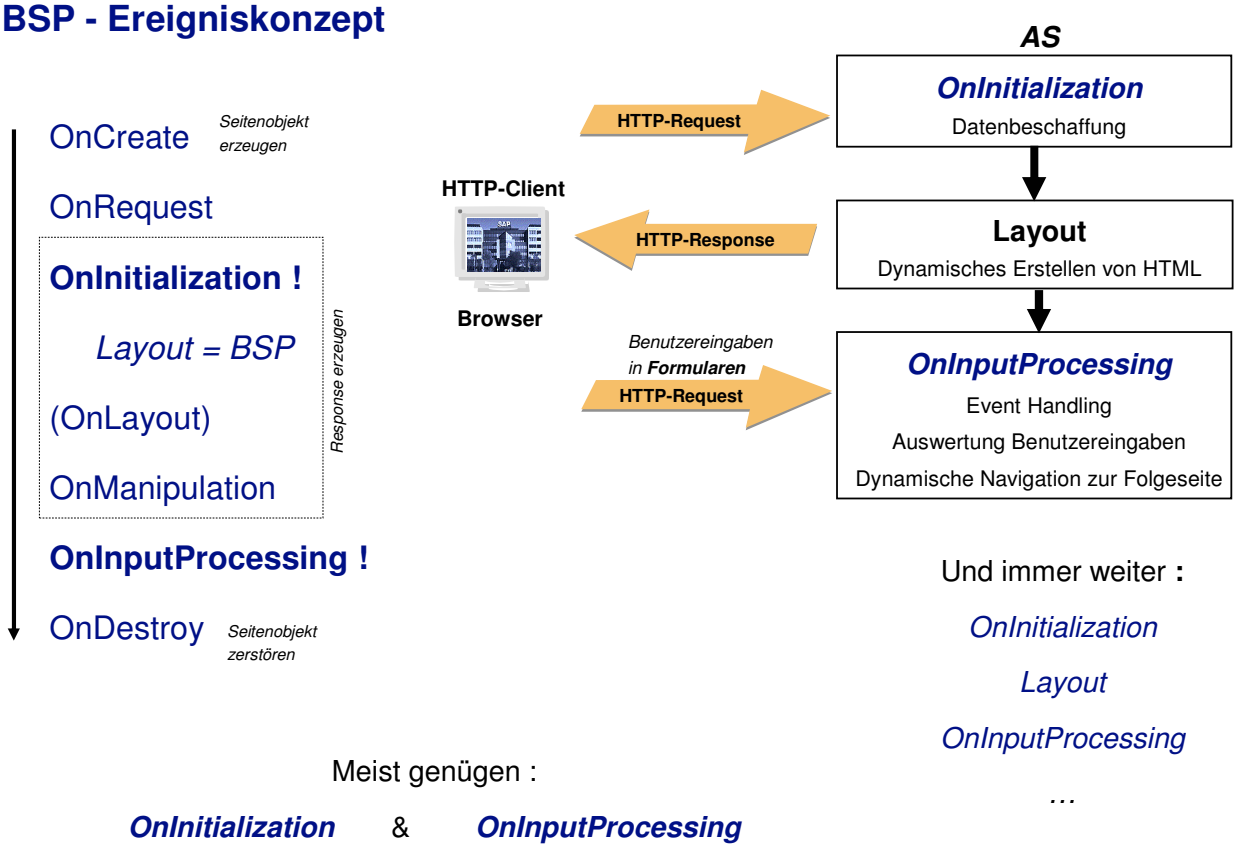
- **OnCreate:** Wird nur beim ersten Aufruf der BSP-Seite durchlaufen, wenn für die Seite noch *kein* **Seitenobjekt** existiert. Dient zum einmaligen Initialisieren von Daten. (Wird bei stateful-Anwendungen nur einmal ausgeführt, da das Seitenobjekt im stateful-Fall nach der Response *nicht* zerstört wird)
- **OnRequest::** Wird bei jedem neuerlichen Zugriff auf die BSP ausgeführt. Dient zum Wiederherstellen interner Datenstrukturen eines Requests.
- **OnInitialization:** Dient der Implementierung der Datenbeschaffung (Lesen von der DB, Füllen interner Tabellen mit DB-Werten, Aufruf von Methoden der Applikationsklasse etc.) Die beschafften Daten sind im Layout der BSP-Seite und anderen Eventhandlern zugreifbar.
- **OnLayout:** Dient dem Aufbau des Layouts, um dann den Response-HTTP-Datenstrom zu erstellen. Ist für Entwickler nicht zugreifbar.
- **OnManipulation:** Dient der endgültigen Manipulation des HTML-Datenstroms. Aufgerufen kurz vor Senden des HTML-Datenstroms, nach Ausführung aller serverseitigen Skripte.
- **OnInputProcessing:** Dieser Event-Handler ist für die Reaktion auf Benutzeraktionen zuständig. Eingabedaten können geprüft und verarbeitet werden. Er muss innerhalb von Formularen im BSP-Layout aufgerufen werden, woraufhin der in ihm enthaltene Programmcode ausgeführt wird. Aus dem Event-Handler heraus kann die Navigation auf folgende BSP-Seiten angestoßen werden. Auch das Weiterleiten von Seitenattribut-Werten ist möglich.
- **OnDestroy:** Nur in *stateless*-Anwendungen durchlaufen. Der Aufruf erfolgt vor dem Löschen des Seitenobjekts. Somit können abschließende Aktivitäten (Datensicherung, Kontext speichern) für die Seite durchgeführt werden. Wird im stateful-Fall *nicht* prozessiert

BSP-Ereigniskonzept : Trennung Layout / Businesslogik



- Die Folie zeigt als **Beispiel** den Eventhandler *OnInitialization*. Über die Auswahlbox kann zwischen den verschiedenen Eventhandlern *gewechselt* werden.
- Gerade auch in den Eventhandlern können **Breakpoints** gesetzt und somit der Debugger aufgerufen werden.

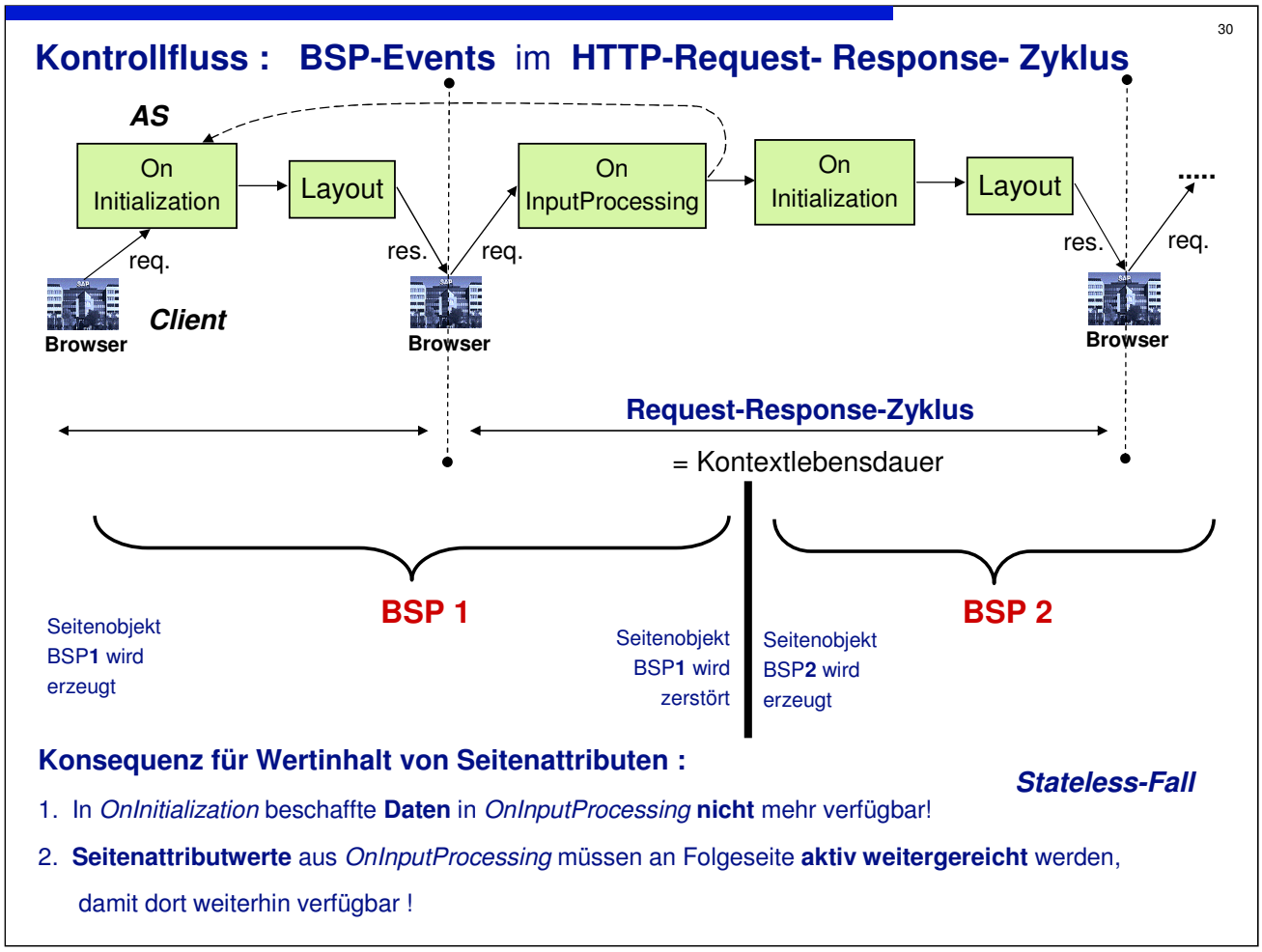
BSP - Ereigniskonzept



- **Eventhandler** dienen dazu, ABAP-Code aus dem BSP-Seiten-Layout *herauszuhalten* und seine Ausführung zu definierten Zeitpunkten und Reihenfolgen zu garantieren. In der BSP-Seite selbst (Layout) kann man sich auf die HTML-Layout-Gestaltung konzentrieren.
- Eine **saubere Entwicklung** sollte streng zwischen **Layout**-Aufbereitung und **Verarbeitungslogik** trennen:

  Scripting für die **statischen Angaben** einer Seite findet in der Layoutverarbeitung statt, ...

  .....während die eigentlichen **dynamischen Verarbeitungsschritte** in den verschiedenen **Eventhandlern** festgelegt werden.



- HTTP ist ein **zustandsloses Protokoll** und bedient **isolierte Request-Response-Zyklen**. Der Browser (Client) fordert an, initiiert den Request / Response-Zyklus. Der Client agiert - der Server reagiert nur und bedient den Request.
- Die Abbildung soll verdeutlichen, dass der Zeitrahmen eines Request-Response-Zyklus **nicht** mit der Prozessdauer einer BSP-Seite übereinstimmt. Vielmehr findet durch die Navigation in *OnInputProcessing* in der Regel ein **Seitenwechsel** statt. Damit die Werte der Seitenattribute der **BSP1** auch in der **BSP2** zur Verfügung stehen, müssen sie deshalb mit dem Aufruf *navaigation→set_parameter*() aktiv an die Folgeseite weitergereicht werden.
- Insbesondere ist zu beachten, dass **Seitenattribut-Werte**, mit denen in *OnInitialization* und im Layout gearbeitet wurde, im stateless-Fall nach Versenden der Response an den Client gelöscht werden - und somit in *OnInputProcessing* der **gleichen Seite** nicht mehr zur Vfg stehen.

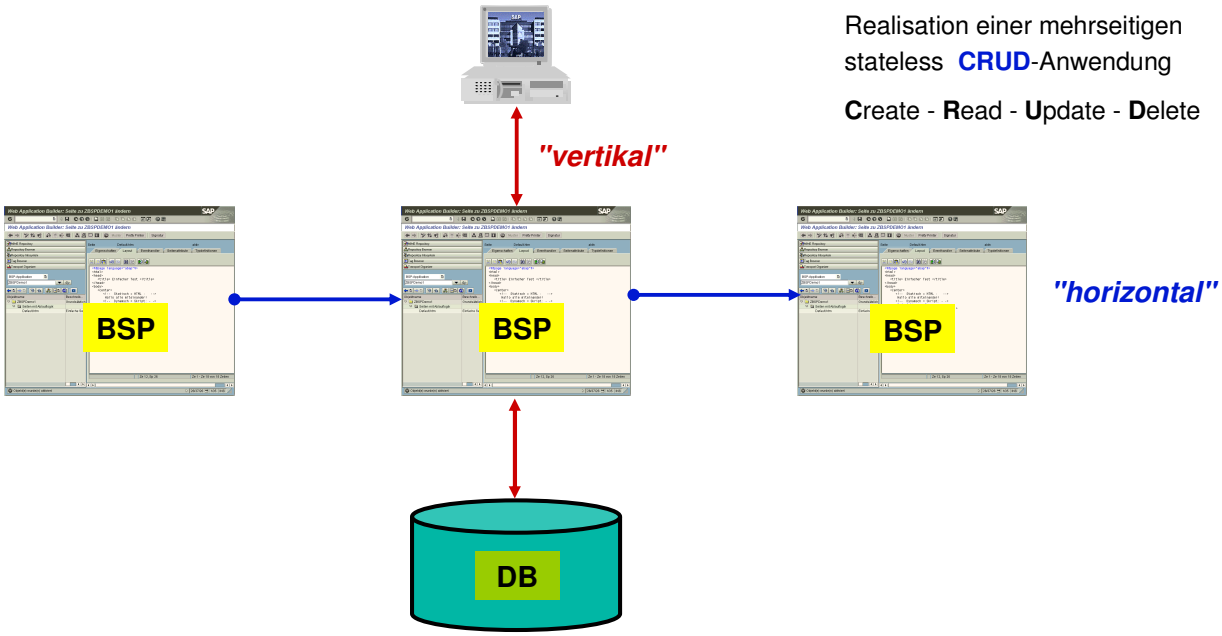
  Der Grund dafür ist, dass im **stateless-Fall** nach dem Versenden der Reponse der Kontext auf dem AS gelöscht wird. Dadurch gehen die **Werte aller Seitenattribute verloren** - und auch **alle Attributwerte des Applikationsklassen-Objekts,** da dieses ebenfalls gelöscht wird.
- Auch beim **Wechsel der BSP-Seite** gehen dort gehaltene Werte verloren. Dies liegt in diesem Fall daran, dass dabei das zugehörige **Seitenobjekt zerstört** wird (**falls** die Anwendung **stateless** läuft) und deshalb die im Seitenkontext gespeicherten Daten gelöscht werden.
- Eine BSP-Seite kann sich auch **selbst aufrufen** und erneut prozessieren. Bei **Selbstaufruf** wird implizite und explizite Navigation unterschieden (s.u.)

Datentransfers in BSP-Anwendungen

Ziel :

Realisation einer mehrseitigen stateless **CRUD**-Anwendung

Create - **R**ead - **U**pdate - **D**elete



Zusammenspiel von **auto-Seitenattributen** und **Eventhandler-Coding** ermöglicht Datentransfer in allen erforderlichen "Richtungen"

- Es soll deutlich werden, dass innerhalb einer BSP-Anwendung zwei **"Richtungen" des Datentransfers** unterschieden werden können :

  Der Datenaustausch mit dem User bzw. der Datenbank.

  Die Übergabe von Daten zwischen einer BSP und der nächstfolgenden BSP

- Für beide "Richtungen" stellen die **Eventhandler** (insbesondere OnInputProcessing) in Zusammenarbeit mit **auto-Seitenattributen** ein technisches Mittel dar.

- **Anmerkung zu HTTP 1.1:**

  Typischerweise enthalten Web-Seiten neben HTML auch viele andere Elemente (z.B. Bilder via IMG-Tag). Um die komplette Web-Seite anzuzeigen, muss der Browser dann für jedes im HTML enthaltene IMG-Tag die entsprechende Bild-Datei durch einen **erneuten** HTTP-Request vom Web-Server anfordern. Da dazu bei HTTP1.0 jedes Mal eine erneute TCP-Verbindung **aufgebaut** werden muss, und da das Auf- und Abbauen nicht ohne Aufwand möglich ist, wurde in HTTP1.1 folgende Verbesserung integriert:

  Wenn der Browser in seiner Anfrage mitteilt, dass er das HTTP1.1-Protokoll unterstützt, dann hält der Server nach dem Senden der Antwort die Verbindung noch einige Sekunden **geöffnet**. In dieser Zeit kann der Browser eine erneute Anfrage senden, **ohne** eine neue Verbindung aufbauen zu müssen.

  Wird die Verbindung für einige Sekunden vom Browser nicht mehr benutzt, dann schließt der Server die Verbindung.

OnInputProcessing

Benutzereingaben : **HTML-Formulare** <FORM> … </FORM>

Absenden : **Eingabeverarbeitung** auf Server

```
... Layout …

<form method = "post" >
<input  type = "TEXT" name = "CUSTOMER" >
<input  type = "SUBMIT"  name = "OnInputProcessing(Opt_1)"  value = "Option 1"  >
<input  type = "SUBMIT"  name = "OnInputProcessing(Opt_2)"  value = "Option 2"  >
</form>
…
```

Namen der *event_id* frei wählbar

Keine spaces zwischen ( ) und event_id !

Submit-Name **muss**(!) *OnInputProcessing* lauten

Verarbeitung von Benutzereingaben in Eventhandler ***OnInputProcessing***

Voraussetzung : BSP-Form hat **SUBMIT**-Inputelement mit **Namen** ***OnInputProcessing***

Auswahl → ruft **Eventhandler** auf

Unterschiedliche serverseitige Auswertungen für ***ein*** Formular durch unterschiedliche **Event_IDs** im Submit-Namen ⇒ Fallunterscheidung im Eventhandler

- Damit Anwender auf HTML-Seiten Daten eingeben können, müssen **Eingabefelder** in einem **HTML-Formular** angeboten werden.Typischerweise werden Eingabefelder, Auswahllisten und Drucktasten angeboten. Der Anwender sendet die Informationen zum Server, der die Daten entgegennimmt, weiterverarbeitet und eine Antwortseite zurückschickt. Eine Webseite kann mehrere Formulare enthalten – diese dürfen nicht geschachtelt werden. Um ein HTML-Formular an den Server senden zu können benötigt man ein **INPUT-Element** vom Typ **SUBMIT**. Dadurch wird eine Drucktaste (Button, Schaltfläche) erzeugt.
- Benutzereingaben werden im **Event *OnInputProcessing*** verarbeitet. Voraussetzung für den Eintritt des Events ist, daß in der BSP ein Formular existiert, in dem mindestens ein INPUT-Element (Button) vom Typ *Submit* definiert ist. Für den Button **muss** der Name *OnInputProcessing* vergeben werden, damit auf dem WAS das Event *OnInputProcessing* ausgelöst wird. Wählt der Anwender die Drucktaste aus, wird das *OnInputProcessing-Event* prozessiert und der darin enthaltene Code ausgeführt.
- Sind in einer Seite **mehrere Buttons** vorhanden, muss der Name des Submit-Buttons zusätzlich mit einer (frei wählbaren) **Kennung** versehen werden. Bei der Ausführung des *OnInputProcessing*-Eventhandlers wird dann das **globale Laufzeitobjekt *event_id*** mit dieser Kennung gefüllt und kann ausgewertet werden. Dies funktioniert auch, wenn mehrere Forms in einer Seite enthalten sind. Wenn **nur eine Alternative** vorgesehen ist, kann auf die Mitgabe der Event-ID *verzichtet* werden – dann genügt "OnInputProcessing" (ohne runde Klammern).
- Um innerhalb *eines* HTML-Formulars zu unterschiedlichen serverseitigen Ausführungen zu gelangen, werden *entsprechend viele* Drucktasten angeboten. Damit der Eventhandler *OnInputProcessing* ausgeführt wird, muß jeder dieser Drucktasten den Namen *OnInputProcessing* tragen. Damit eine **Fallunterscheidung** möglich wird, steht das globale Objekt **event_id** im Eventhandler zur Vfg. Jeder Drucktaste wird eine andere Event_ID zugeordnet, die im Eventhandler *OnInputProcessing* abgefragt werden kann.
- Damit der BSP-Eventhandler-Mechanismus funktioniert, darf in der Form **kein action-Attribut** angegeben werden!

Ablaufsteuerung im Handler ***OnInputProcessing***

```
<form method="post" >                                   Layout
  <input  type = "SUBMIT" name = "OnInputProcessing(Opt_1)"  value = "Mitarbeiter" >
  <input  type = "SUBMIT" name = "OnInputProcessing(Opt_2)"  value = "Administrator" >
</form>
…
```

```
*ABAP-Code :        OnInputProcessing
…
CASE  event_id.
  WHEN  'Opt_1'.
    * Anmeldung für Mitarbeiter :  (URL)
     navigation→goto_page( 'start_ma.htm' ).
  WHEN  'Opt_2'.
    * Anmeldung für Admin :  (Nav.request)
     navigation→next_page( 'start_adm' ).
ENDCASE.
…
```

Fallunterscheidung mittels ***event_id***

Dynamische Folgeseite-Navigation mittels globalem Objekt ***navigation*** :

Methoden zum **Folgeseiten-Aufruf**.

* **Beenden** einer BSP-Session +

* **Freigabe** von **stateful** Server-Ressourcen :

navigation→exit('end.html').

- Die ***Event_ID*** steht zum Zeitpunkt **OnInputProcessing** zur Fallunterscheidung in einer Case-Struktur zur Vfg. Auch die dynamische Navigation zur **Folgeseite** wird im BSP-Modell typischerweise im Eventhandler *OnInputProcessing* gesteuert:
- Um eine BSP-Anwendung in ihrem **Ablauf dynamisch** gestalten zu können (**explizite** Bestimmung der Folgeseite), steht das globale, automatisch instantiierte **Objekt** ***navigation*** zur Vfg. (ICF-Interface IF_BSP_NAVIGATION, implementiert in CL_BSP_NAVIGATION.) Es gibt **Methoden** zum Aufruf **der Folgeseite**.

  navigation→exit(url) :

  - Beendet die aktuelle *Session* der laufenden Anwendung und verzeigt auf die angegebene URL. Eine BSP-Anwendung kann somit *neu gestartet* werden, indem mit *exit*( ) wieder zur Einstiegsseite der Anwendung gesprungen wird. Dadurch werden durch die aktuelle Session auf dem Server gehaltene Ressourcen freigegeben (zB offene TCP/IP-Verbindungen). Eine BSP-Applikation sollte deshalb mit einem exit() zu einer Abschlusseite enden.

  navigation→ goto_page('test.htm') :

  - Parameter ist direkt übergebene **URL** der Folgeseite; angegebene BSP-Seite wird aufgerufen.

  navigation→ next_page('start_seite') :

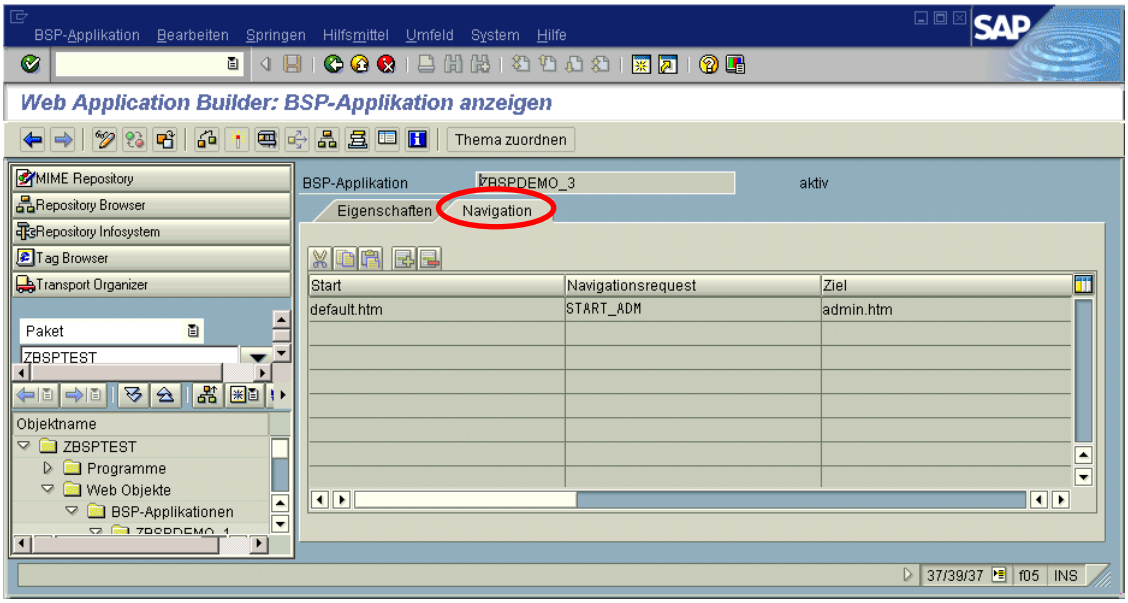
  - Es wird ein **Navigationsrequest** übergeben, die zugehörige BSP-Seite (gemäß definierter **Navigationsstruktur**) wird aufgerufen.
- **Redirect:** Zum Zeitpunkt **OnInitialization** steht das automatisch instantiierte ***response***-Objekt zur Verfügung, dessen Methode redirect() einen Redirect auf eine andere Seite veranlasst. Somit kann der Server Anfragen automatisch zur korrekten Seite weiterleiten.

  response→redirect('http://www.sap.com') .

Navigationsstruktur BSP-Anwendung / Navigationsrequest

Globale Definition über **BSP-Anwendung** → Tab ***Navigation***

- Navigationsstruktur *ohne Änderungen im Code* der BSPs änderbar ⇒
- **Trennung von Navigationsstruktur & BSP-Code + Überblick**



- Bei Erstellung einer BSP-Anwendung kann man die **Navigationsstruktur** (Abfolge der Seiten) festlegen. Im Reiter *Navigation* der *Eigenschaftsseite* der BSP-Anwendung können Navigationsrequests deklariert werden.

  Ein **Navigationsrequest** wird durch Angabe seines Namens sowie der zugehörigen Start- und Zielseite definiert. Auf diese Weise kann die Navigationsstruktur später ohne Änderungen im Code geändert werden kann. Navigationsstruktur und BSP-Quellcode sind somit getrennt.

  Trick:

  Man kann sogar als ***event_id*** einen Navigationsrequest übergeben und direkt an die Methode *next_page()* weiterreichen:

  - Im Layout:

    *<input type="submit" name="OnInputProcessing(**start_adm**)" value="Admin" >*

  - In OnInputProcessing:

    *navigation→ next_page(**event_id**).*

- **Anmerkung** zum Aufruf von OnInputProcessing:

  Der beteiligten **HTML-Form** darf **kein action-Attribut** mitgegeben werden – sonst landet man direkt auf einer anderen Seite bzw. OnInputProcessing wird ***nicht*** durchlaufen.

Explizite Wertabfrage von Formularfelder-Werten (low-level)

```
<form method="post" >                                          Layout
  <input  type = "text"  name = "author"   >
  <input  type = "SUBMIT" name = "OnInputProcessing(Opt)"  value = "Eingabe" >
</form>
```

```
*ABAP-Code :                                   OnInputProcessing
DATA :  feldwert  TYPE  STRING.

feldwert = request→get_form_field( 'author' ).
…
CASE event_id.
   WHEN 'Opt'.
        …
ENDCASE.
…
```

Business-Logik in Abhängigkeit von Benutzereingaben

Durch **LZ** instanziiertes Objekt ***request*** mit Methode :

get_form_field( )

Wert Formularfeld kann in Eventhandlern abgefragt / ausgelesen werden.

Anwendbar auf **alle Arten** von HTML-Eingabeelementen.

Low-Level-Zugriff via

ICF-Objekt

- Der **Wert eines Formularfeldes** kann in den Eventhandlern **explizit abgefragt** und in eine ABAP-Variable ausgelesen werden.
- Hierzu steht das globale, automatisch instanziierte Laufzeitobjekt ***request*** zur Vfg. (vom Typ **ICF_HTTP_REQUEST**).
- Auf diese Weise kann nicht nur der Wert von Textfeldern, sondern auch von Formularfeldern anderen Typs (radio, check, select, ...) abgefragt werden.

Automatisches Füllen von Seitenattributen – Auto-Seitenattribute

```
<form method="post" >                                   Layout
  <input  type = "text"   name = "customer "   >
  <input  type = "SUBMIT" name = "OnInputProcessing"  value = "Eingabe" >
</form>
```

```
*ABAP-Code:      OnInputProcessing

DATA:  test  TYPE STRING .
test = customer .
…
…
```

```
* Seitenattribute
customer   TYPE  STRING.     AUTO
```

Seitenattribut ***customer*** gefüllt
mit eingetragenem *Formularfeld-Wert*

Auto-Seitenattribute sind Import-Schnittstelle einer BSP-Seite :

BSP-Seitenattribut in *OnInputProcessing* **automatisch** mit Wert des ***gleichnamigen*** Request-Formularfeldes gefüllt.

Muss **Auto-Seitenattribut** sein !

Leistung BSP-Framework - Data-Binding

Erspart (evtl. zahlreiche) explizite Aufrufe request→get_form_field( )

- **Auto-Seitenattribute** stellen die Import-**Schnittstelle einer BSP** dar. Für die Füllung von Seitenattributen **derselben** BSP mit Formularwerten gilt das Gleiche wie für die Übergabe von Formularwerten an Seitenattribute von Folge-BSPs (s.u.):
- Die **aktuelle** BSP-Seite kann auf die Inhalte der **Eingabefelder einer Form** zugreifen. Es ist **Namensgleichheit** erforderlich: Der **Name** des Eingabefeldes im Seitenlayout der aktuellen Seite muss **identisch** sein mit dem Namen des Seitenattributs auf der aktuellen BSP.
- Damit ein Seitenattribut von außen gefüllt werden kann, muss es also als **Auto-Seitenattribut** deklariert werden. Zu einem Name-Value-Paar des Eingabeformfelds muss ein **gleichnamiges** Auto-Seitenattribut in der Seite existieren, damit dieses automatisch mit den Eingaben des Benutzers gefüllt werden kann. Es handelt sich um einen **Data-Binding-Mechanismus**: Durch das kontrollierende Framework werden Variablen automatisch zum richtigen Zeitpunkt mit Werten versorgt.
- Durch **Auto-Seitenattribute** erspart man sich die explizite Abfrage der Formularwerte mittels des **request-Objekts**. Sind entsprechende Auto-Seitenattribute vorhanden, so werden diese **automatisch** mit den Benutzereingaben der Form gefüllt.
- Über **request→get_form_field()** hat man jedoch natürlich weiterhin stets Zugriff auf alle Formularwerte – auch wenn für diese **kein** gleichnamiges Seitenattribut deklariert wurde.

Datentransfer zwischen **verschiedenen** BSPs – **Auto-Seitenattribute**

```
<form method="post" >                                          Layout
  <input  type = "text"  name = "city"   >
  <input  type = "SUBMIT" name = "OnInputProcessing"  value = "Eingabe" >
</form>
```

```
*ABAP-Code:        OnInputProcessing
…
 * Übertragen des Wertes  :
navigation→set_parameter( 'city' ) .

navigation→goto_page( 'results.html' ).
…
```

Anmerkung :

Alle Bezeichner sind *nicht* case sensitive

Von *Vorgängerseite* mit Inhalt ***füllbar*** :

navigation→set_parameter( )

Voraussetzung :

Namensgleichheit der *Eingabefelder* der **Vorgänger-BSP** und der *Auto-Seitenattribute* der **Nachfolge-BSP.**

Methode *set_parameter( )* sendet Wert des **Eingabefeldes *city*** an **gleichnamiges Auto-Seitenattribut *city*** der **Folgeseite** *results.htm*

Dort in *OnInitialization* und *Layout* zur Vfg.

- Seitenattribute dienen zur Definition von Variablen, die zu allen Zeitpunkten innerhalb der BSP-Seite sichtbar sind. Dabei stellen **Auto-Seitenattribute** auch die **Schnittstelle einer BSP** dar. Auto-Seitenattribute sollten in der Regel vom Typ *String* sein, damit bei Übertragung keine Konvertierungsfehler auftreten !
- Sowohl die **aktuelle** als auch die **gerufene** BSP-Seite können auf die Inhalte der Eingabefelder einer Form zugreifen. Dabei muß der Name des Eingabefeldes im Seitenlayout der aktuellen Seite identisch sein mit dem Namen des zu füllenden Seitenattributs auf der Folge-BSP.
- Damit ein Seitenattribut von außen gefüllt werden kann, muss es als **Auto-Seitenattribut** deklariert werden. Das Übertragen von Feldinhalten einer BSP an ihre Folge-BSP beruht auf **Namensgleichheit** der Variablen:

  Zu einem Name-Value-Paar des Eingabeformfelds der *Vorgängerseite* muss ein **gleichnamiges** Auto-Seitenattribut in der *Folgeseite* existieren.
- Der **Wert** des Name-Value-Paares kann auf verschiedene Weise an gleichnamige Auto-Seitenattribute der Folgeseite übermittelt werden:

  1. In den Parametern eines Links
  2. In den Formulardaten
  3. Aus dem Event **OnInputProcessing** heraus mittels des ***navigation-Objekts***:

     navigation→ set_parameter(' feldname ').
- Diese einfache Form dieses Methodenaufrufs funktioniert ***nur*** bei **Namensgleichheit**! Das Coding ist eine *abgekürzte* Form der *generell* anwendbaren Aufruffolge :
  - *wert =* ***request→get_form_field('formfeldname').***
  - ***navigation→set_parameter(name='seitenattributname' value = wert).***
- In der ausführlichen Form können **auch die Inhalte ganzer interner Tabellen** an die Folgeseite übertragen werden.

Datentransfer zwischen **verschiedenen** BSPs **ohne** Namensgleichheit

```
<form method="post" >                                          Layout
  <input  type = "text"  name = "author"   >
  <input  type = "SUBMIT" name = "OnInputProcessing"  value = "Eingabe" >
</form>
```

```
* OnInputProcessing
DATA :  feldwert  TYPE  STRING.
DATA :  info  TYPE STRING  value  'Irgendwas'.
feldwert  =  request→get_form_field( 'author' ).

navigation→set_parameter(
                    name = 'aut'   value = feldwert  ) .
navigation→set_parameter(
                    name = 'information'   value = info  ) .

navigation →goto_page( 'next.htm' ).
```

Übergabe von Formularwerten an Folgeseite mit ***nicht*** *gleichnamigen* Auto-Seitenattribut.

Dadurch **auch** Daten an Folgeseite übergebbar, …

die **keinen** Benutzereingaben entsprechen !

- Der **Wert eines Formularfeldes** kann in den Eventhandlern **explizit abgefragt** und in eine ABAP-Variable manuell ausgelesen werden. Hierzu steht das globale Laufzeitobjekt ***request*** zur Vfg.
- Auf diese Weise lässt sich auch der **ausführliche Aufruf** der Methode *navigation→ set_parameter*() durchführen – der **erforderlich** ist, wenn der **Formularfeldname** und der **Name des Auto-Seitenattributs** auf der Zielseite **nicht übereinstimmen** !
- Natürlich können durch den ausführlichen Aufruf der Methode auch Daten an die Folgeseite gesendet werden, die der User nicht zuvor eingegeben hat. Auf diese Weise kann eine BSP-Seite an eine andere Seite **beliebige** Daten schicken, für die auf der empfangenden Seite ein Auto-Seitenattribut existiert.

Möglichkeiten Datentransfer

default.htm + **next.htm** haben **auto-Seitenattribut customer** Typ String

Code sei Teil von default.htm

```
<form action="next.htm" method="post">
    <input type="text" name="customer" value="Meier11">
    <input type="SUBMIT" name="Send" value="Form">
</form>
```

OnInitialization von next.htm

Man landet in ...

```
<form action="default.htm" method="post">
    <input type="text" name="customer" value="Meier22">
    <input type="SUBMIT" name="Send" value="Form">
</form>
```

OnInitialization von default.htm

```
<form method="post">
    <input type="text" name="customer" value="Meier33">
    <input type="SUBMIT" name="OnInputProcessing" value="Form">
</form>
```

OnInputProcessing von default.htm

OnInitialization von next.htm

```
<a href="next.htm?customer=Meier44"> Link auf Folgeseite mit Name-Wert-Paar </a>
```

OnInitialization von default.htm

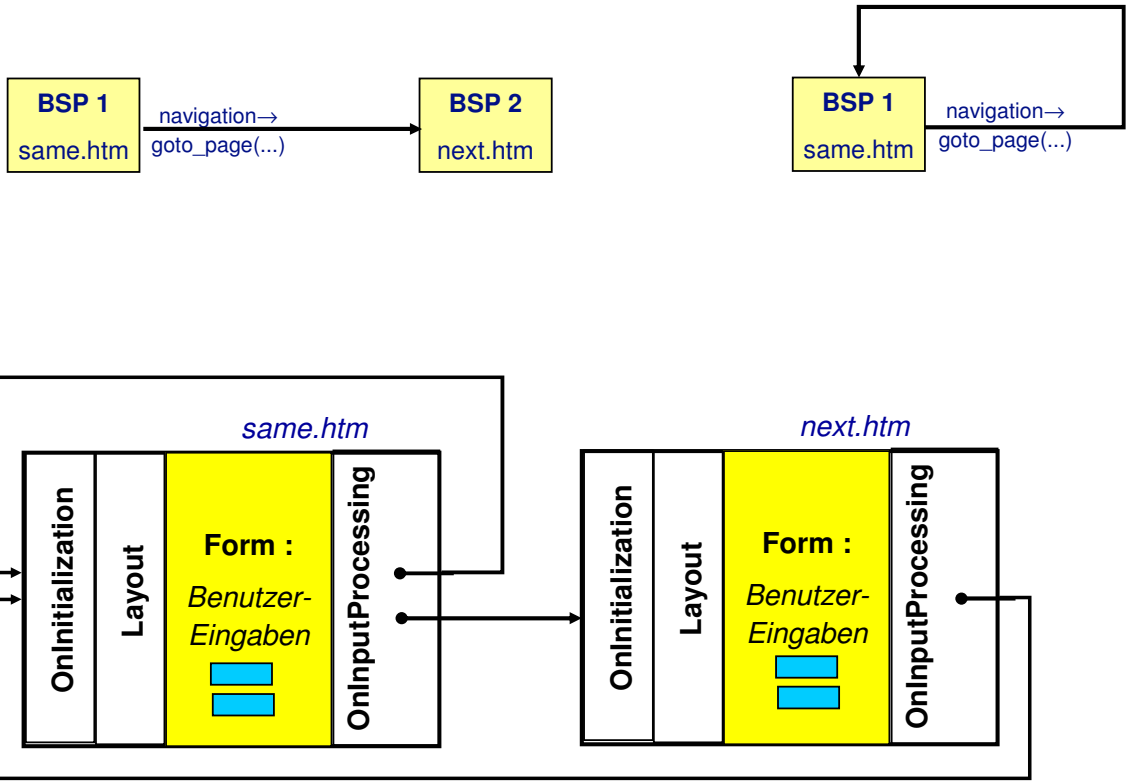
```
<a href="default.htm?customer=Meier55"> Link auf selbe Seite mit Name-Wert-Paar </a>
```

OnInputProcessing von default.htm

```
<a href="?OnInputProcessing(opt1)&customer=Meier66"> Link auf OnInputProcessing </a>
```

- Es wird ein **Überblick** gegeben über die Möglichkeiten, **vom Frontend** bei einem Request Daten an den **Server** zu übergeben.
- Dafür stehen verschiedene Möglichkeuten der Navigation und des Datentransfer durch **Formulare** und **Links** zur Verfügung.
- Grundsätzlich kann **dieselbe** BSP-Seite oder eine **andere** BSP-Seite angesprungen und mit Werten versorgt werden. Dabei ist zu beachten, welcher Eventhandler erreicht werden kann. Am flexibelsten sind die Möglichkeiten, die sich bei Nutzung des Events **OnInputProcessing** derselben BSP-Seite bieten - denn in diesem Eventhandler kann eine weitere Aufarbeitung der eingegebenen Daten vorgenommen werden und **wahlweise** ein Transfer zu einer anderen BSP-Seite vorgenommen werden.
- Im Paket **ZBSPTest** fasst die Demonwendung **ZBSPDataTrans** die Möglichkeiten zusammen.

Explizite Navigation auf **selbe** ***oder*** **andere BSP**



- Es gibt folgende **Fälle unterschiedlicher Lebensdauer** von automatischen und nicht-automatischen Seitenattributen bei der Navigation zwischen Seiten:

  Lebensdauer von Attributen bei **Verbleiben** auf **der selben** Seite.

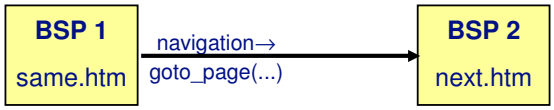
  Lebensdauer von Attributen bei der **Navigation** auf die **selbe** Seite.

  Lebensdauer von Attributen bei der **Navigation** auf eine **andere** Seite.

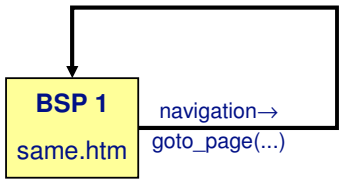
Explizite Navigation auf selbe *oder* andere BSP

Expliziter *navigation*-Aufruf :

```
navigation→goto_page( 'next.htm' ).
navigation→next_page( 'TONEXT' ).
navigation→goto_page( 'same.htm' ).
navigation→next_page( 'TOSAME' ).
```



... auf ***selbe*** BSP ***zurück*** **oder** auf ***andere*** BSP



Konsequenz : Seitenobjekt stets neu instanziiert !

⇒ **Explizite Seitenattribut-Wertübergabe erforderlich, …**

… wenn Seitenattribut-Werte erhalten bleiben soll :

Seitenattributwerte mit **navigation→set_parameter( )**

an gleichnamige Auto-Seitenattribute übergeben,

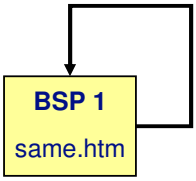
um gesetzte Werte **nicht zu verlieren** !

- Durch Aufruf von *navigation→goto_page*() und *navigation→next_page*() wird **explizit** navigiert. Hierbei spielt es keine Rolle, ob auf **dieselbe** Seite navigiert wird, von der man kommt, oder ob auf eine **andere** Seite.
- Bei **expliziter Navigation** wird das Seitenobjekt immer **neu instantiiert**. Hier müssen sowohl im stateful als auch im stateless Fall die Attribute mit *navigation→set_parameter*( ) an gleichnamige ***auto***-Seitenattribute übergeben werden, um die gesetzten Werte **nicht** zu verlieren. Andernfalls sind die Werte der entsprechenden Seitenattribute wieder initial!

Implizite Navigation auf selbe BSP

Layout

```
<form method="post" >
  <input  type = "SUBMIT" name = "OnInputProcessing(Same)"  value = "Nochmal" >
  <input  type = "SUBMIT" name = "OnInputProcessing(Next)"  value = "Weiter" >
</form>
```



OnInputProcessing

```
CASE event_id.
  WHEN 'Same' .
*   keine explizite Navigation  !
  WHEN 'Next'.
    navigation→goto_page( 'next.htm' ) .
ENDCASE.
```

Keine explizite Navigation im Eventhandler ⇒

Die selbe Seite wird **nochmals** durchlaufen,

OnInitialization erneut prozessiert :

Im **selben** Request-Response-Zyklus !

Seitenattributswerte *bleiben* erhalten !

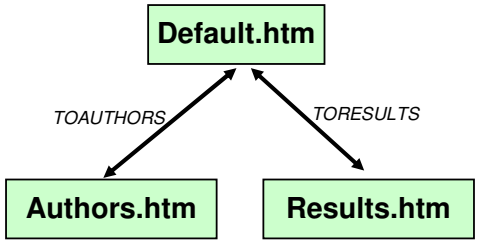
Egal ob stateful oder stateless

- Durch Benutzereingabe wird ein Request abgesetzt. Da **keine explizite Navigation** angegeben ist, wird **wieder die selbe Seite durchlaufen**. Es findet keine Neu-Instanziierung des Seitenobjekts statt.
- In diesem Fall werden die Werte der Seitenattribute sowohl im stateful als auch im stateless Fall automatisch von *OnInputProcessing* nach *OnInitialization* übernommen.
- Dies gilt sowohl für **automatische** als auch **nicht-automatische** Seitenattribute.
- Durch die **Formulareingaben** des Requests werden gleichnamige automatische-Seitenattribute neu aus dem Request mit den Eingabewerten gefüllt.
- Die tabellarische Übersicht zeigt, dass das Verhalten einer BSP-Anwendung sich deutlich **unterscheidet** - je nachdem, ob explizite oder implizite Navigation verwendet wird.

Beispielanwendung

Navigation

Start	Navigationsrequest	Ziel
default.htm	**TOAUTHORS**	authors.htm
default.htm	**TORESULTS**	results.htm



BSP-Seiten:

Default.htm = Einstiegsseite : Layout + OnInputProcessing

Authors.htm = Alle Autoren : Layout + Seitenattribute + Typdefinitionen + OnInitialization

Results.htm = Bücher zu Autor : Layout + Seitenattribute + Typdefinitionen + OnInitialization

DB-Tabellen :

ZBSBOOKS

isbn char13 | *title* char132 | *publisher* char40 | *publyear* numc 4

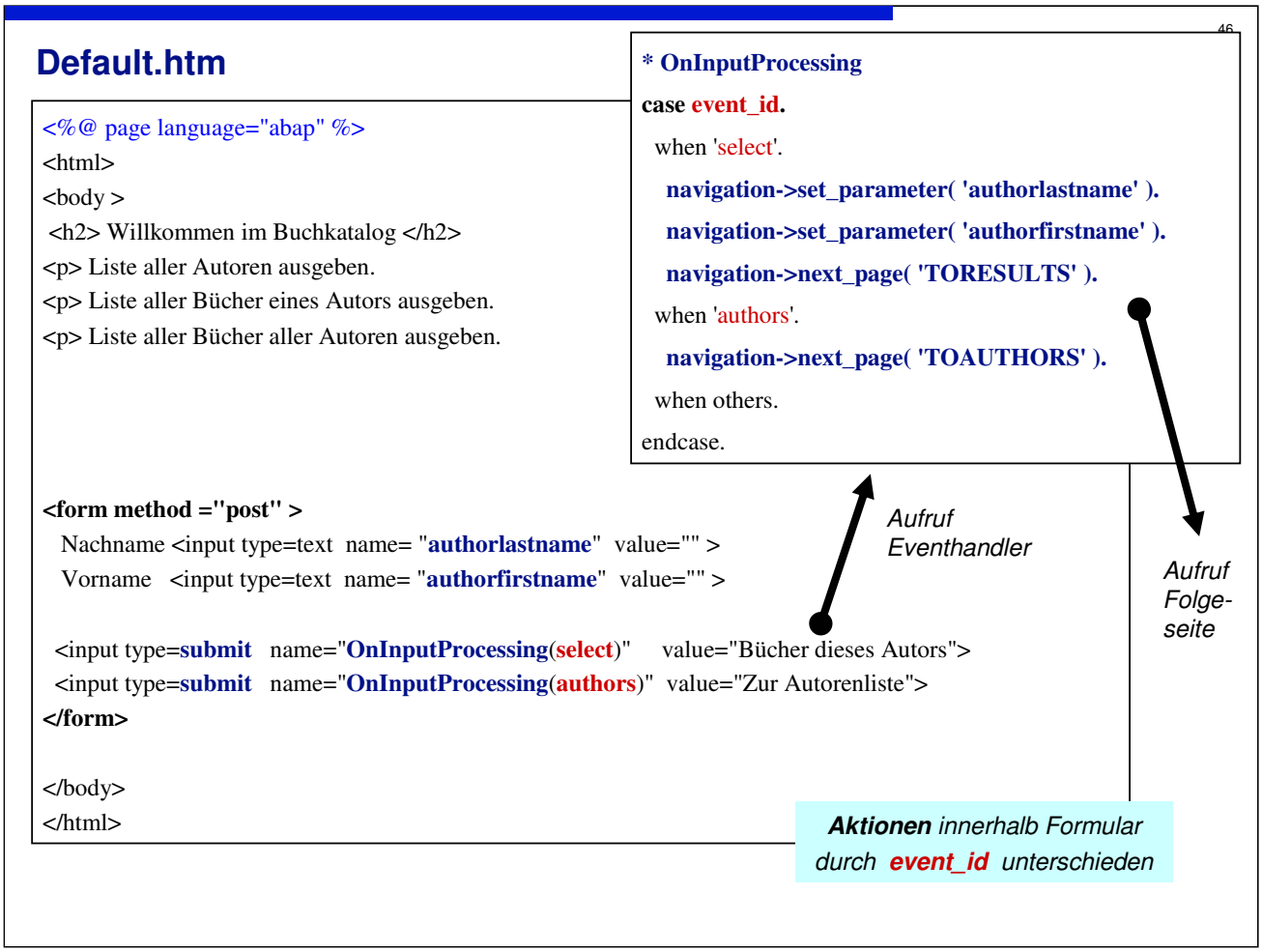
ZBSAUTHORS

isbn char13 | *authlname* char40 | *authfname* char40

Einfache Demonstration wesentlicher BSP-Elemente (stateless)

keine Anwendungsklasse, keine Includes, kein JS / CSS …

- Die BSP-Anwendung befindet sich im Paket **ZBSPTEST** in **ZBSPBOOKS**.



- Der Einstieg in die BSP-Anwendung geschieht über die Seite **Default.htm**. Deren *Layout* wird prozessiert und dem User präsentiert. Der User kann sodann in der enthaltenen *Form* Eingaben vornehmen. Die Eingaben werden dann im Eventhandler *OnInputProcessing* verarbeitet.
- Im **Eventhandler** findet die Navigation zur Folgeseite mittels Objekt *navigation* statt. Dessen Methode *next_page*() verwendet die definierten Navigationsrequests. An die Seite *results.htm* werden die Parameter *authorlastname* und *authorfirstname* übergeben.
- Die **einfache Form** des *set_parameter()*-Aufrufs ist möglich, da die Namen der Seitenattribute *authorfirstname* und *authorlastname* auf allen Seiten der Anwendung und in allen Formularen gleich lauten.
- Anmerkung: Wenn auf einer BSP-Seite irgendwelche **Benutzeraktionen** passieren, für die **keine Navigation** zu einer anderen Seite vorgesehen ist, dann wird die *selbe Seite* einfach *nochmals* an den Client gesendet (die Seite "bleibt offen"). Dies kann man z.B. ausnutzen, um dem Benutzer bei fehlerhaften Eingaben die Gelegenheit zur Korrektur zu geben.

Authors.htm

```
<%@ page language="abap" %>
<html>
<body>
 <h2> Liste der Autoren </h2>

 <table border=1>
  <tr>  <th><b>  Vorname    </b></th>
        <th><b>  Nachanme  </b></th>
  </tr>

<% data:  wa_author type zbsauthors.
     loop at authors into wa_author. %>
  <tr>
   <td> <%= wa_author-authfirstname %> </td>
   <td> <%= wa_author-authlastname %> </td>
  </tr>
<% endloop.  %>

 </table>
 <a href = default.htm>  Zum Einstieg  </a>
</body>
</html>
```

*** Typdefinitionen**

types: **zauthortab_type** type standard table of zbsauthors.

*** Seitenattribute**

authors type zauthortab_type.

*** OnInitialization**

select * from **zbsauthors** into table **authors**.

sort authors by authlastname.

delete adjacent duplicates from authors
comparing authlastname authfirstname.

Prozessieren
Layout

Businesslogik :
DB-Zugriff
+
Datenaufbereitung

- Die Seite verwendet selbstdefinierte **Tabellentypen** und verfügt über **Seitenattribute**. Das Seitenattribut ***authors*** ist eine **interne** Tabelle.
- Der **Eventhandler *OnInitialization*** wird vor dem Layout prozessiert und führt eine **DB-Abfrage** durch. Dadurch wird die interne Tabelle ***authors*** mit Werten gefüllt.
- Im **Layout** werden die Daten der internen Tabelle *authors* auf der Seite ausgegeben. Es gibt keine Benutzereingaben - nur die statische Navigation zurück zu *default.htm* ist möglich.

Results.htm

```
* Typdefinitionen
types: zauthortab_type   type  standard table of  zbsauthors.
types: zbooktab_type     type  standard table of  zbsbooks.
```

```
* Seitenattribute
authorlastname   type  STRING.   Auto
authorfirstname  type  STRING.   Auto

authors   type   zauthortab_type.
books     type   zbooktab_type.
```

```
* OnInitialization
select * from zbsauthors into table authors.

if authorlastname is initial.
      authorlastname = '*'.
endif.
if authorfirstname is initial.
       authorfirstname = '*'.
endif.
delete authors where not authlname cp authorlastname or not authfname cp authorfirstname.

select  *  from zbsbooks  into table  books.
```

Businesslogik :
DB-Zugriff
+
Datenaufbereitung

Prozessieren Layout … →

Nur passende Einträge bleiben in interner Tabelle

- Beim Aufruf der Seite **Results.htm** werden die **Auto-Seitenattribute** mit Werten gefüllt, die aus der Vorgängerseite **Default.htm** durch den Aufruf von *navigation→set_parameter()* übergeben wurden.
- Die Seite verwendet selbstdefinierte **Tabellentypen** und verfügt über **Seitenattribute**. Die Seitenattribute *authors* und *books* sind interne Tabellen.
- Der **Eventhandler *OnInitialization*** wird vor dem *Layout* prozessiert und führt eine DB-Abfrage durch. Dadurch werden die interne Tabelle *authors* und *books* mit Werten gefüllt.
- Im Coding der Eventhandler haben wir auf **Fehlerprüfungen** verzichtet. Natürlich wäre es besser, bei kritischen Statements wie DB-Zugriffen auf Fehler zu prüfen durch Abfrage des **sy-subrc**-Wertes. In diesem Fall könnte auf eine **Fehlerseite** verzweigt und eine Fehlermeldung mitgegeben werden :

```
if  sy-subrc  <>  0.
  DATA: status type  string  value  ' Fehler bei DB-Zugriff '.
  navigation→set_parameter( 'status' ).        navigation→goto( 'error.htm' ).
endif.
```

- Die **Fehlerseite *error.htm*** könnte ein Auto-Seitenattribut *status* vom Typ String enthalten und würde dessen Wert im Layout präsentieren:

```
<%@ page language="abap" %>
<html>
   <title> Fehler </title>
   <body>
         Fehler aufgeteten:   <%= status %>    <p>
        <a  href="default.htm">     Zurück zum Einstieg    </a>
</body>
</html>
```

Results.htm (*Layout*)

```
<%@ page language="abap" %>
<html>
<body>
 <h2> Ergebnis </h2>
<% if authors is initial.%>
  <h3> Keine Treffer für:  <%= authorlastname %>, <%= authorfirstname %>.  </h3>
<% else. %>
  <h3> Treffer :  </h3>
  <table border=1>
    <tr>  <td> ISBN   </td>  <td> Title  </td> <td> Verlag  </td>  <td> Erscheinungsjahr  </td>  </tr>
<%  data:    wa_book type zbsbooks,     wa_author type zbsauthors.
       loop  at  authors  into  wa_author.
       loop  at  books     into  wa_book   where  isbn = wa_author-isbn.   %>
   <tr>
    <td>  <%= wa_book-isbn    %> </td>          <td>  <%= wa_book-title    %>         </td>
    <td>  <%= wa_book-publisher %> </td>     <td>  <%= wa_book-publyear %>   </td>
   </tr>
<% endloop.    endloop. %>
  </table>
<% endif. %>
  <a href = default.htm>    Zurück zum Einstieg     </a>
</body>
</html>
```

- Im **Layout** werden diejenigen Daten der internen Tabelle ***books*** auf der Seite ausgegeben, die zu des ISBN-Werten des gewünschten Autors passen.
- Es sind keine Benutzereingaben vorgesehen - nur die statische Navigation zurück zu *default.htm* ist möglich.

- In der Anwendung haben wir auf **Seitenfragmente** verzichtet. Um ein einheitliches Layout zu erreichen, könnte es sinnvoll sein, ein passendes Seitenfragment zu definieren und in allen Seiten zu inkludieren. Ein solches Seitenfragement zum Inkludieren am Beginn jeder BSP könnte wie folgt aussehen:

  Seitenfragment *head.htm* zu inkludieren mit: <%@ include file="head.htm" %>

  ```
  <%@ page language="abap" %>
  <html>
  <head>     <title>  Mini-Bookshop </title>     </head>
  <body>
  <h2>   <img height="80" width="80" src="logo.jpg">  Bookshop   </h2>
  <hr>
  ```

  Das **Seitenfragment enthält keine schließenden </body> und </html>-Tags**, da es am Anfang der BSPs inkludiert werden soll. Entsprechend dürften die inkludierenden BSPs keine öffnenden <body> und <html>-Tags enthalten.

Businesslogik : Anwendungsklasse

* Verwendung globales *Application-Object* :

application→get_customer_data (

exporting name = 'Schmidt'

changing customer_data = itab_data).

Globales Objekt *application* :

Zugriff auf Methoden + Attribute der Anwendungsklasse in allen Events und Layout aller BSPs einer Anwendung.

In public **Methoden** der **Anwendungsklasse Businesslogik konzentrieren** !

Globale ABAP-Klasse :

- Beliebig viele **Methoden** + Attribute
- Paramterloser Konstruktor

Trennung Präsentationslogik (Layout) und Anwendungslogik (Anwendungsklasse)

Anwendungsklasse in ***Eigenschaften*** der **BSP-Anwendung** zugeordnet.

z.B.: Klasse *ZCL_CUSTOMER* mit public Methode *get_customer_data*( )

Über *application-Objekt* **aufrufbar**.

In ***Eigenschaften*** der BSP-Applikation unter ***Anwendungsklasse*** Klassennamen eintragen.

Durch **Doppelklick** wird Anlege-Sequenz durchlaufen + **ClassBuilder** aufgerufen.

- Mit der globalenObjekt-Referenz ***application*** kann man auf Methoden und Attribute der zugeordneten Anwendungsklasse zugreifen. Die *application*-Referenz steht ***jeder* Seite** einer BSP-Anwendung mit Anwendungsklasse zur Vfg und wird beim Start der BSP-Anwendung *automatisch* instantiiert. Eine BSP-Applikation darf nur eine Applikationsklasse beinhalten, wohingegen eine Applikationsklasse mehreren Applikationen zugeordnet werden kann.
- Um identisches Coding (Methoden) in *verschiedenen* Eventhandlern nutzen zu können (ohne es zu duplizieren), kann es **global** in einer ABAP-Objects Klasse abgelegt werden.
- Es handelt sich um ABAP-Coding, in dem **Businesslogik** abgelegt ist - also derjenige Teil der BSP-Anwendung, der nichts mit der Gestaltung des Layouts (optische Erscheinung) zu tun hat, sondern Datenverarbeitung und Datenbeschaffung erledigt.
- Das globale Objekt *APPLICATION* steht in **allen Eventhandlern** zur Vfg. Somit kann in jedem Eventhandler ABAP-Coding aus der Anwendungsklasse aufgerufen werden. Ziel ist es, Layout und Businesslogik möglichst zu trennen. Zu vermeiden ist stets eine Überfrachtung des *Layouts* mit Businesslogik.
- Die BSP-Architektur **nähert** sich so dem **Model-View-Controler (MVC) –Paradigma**: Je komplexer die Webapplikation, desto wichtiger ist es zu vermeiden, daß die Geschäftslogik zusammen mit Präsentationslogik und statischem HTML auf kaum wartbare Art in einer Vielzahl von Webseiten verstreut ist.
- **Inhalte der Anwendungsklasse**:

  Typischerweise Methoden, z.B. zur Datenbeschaffung oder Datenprüfung.

  Attribute, z.B. Daten in internen Tabellen, um wiederholte DB-Zugriffe zu vermeiden. Dies **funktioniert jedoch nur bei stateful-Anwendungen.**

Ausgabeaufbereitung : page - Objekt

Klasse CL_BSP_PAGE

Interne Darstellung von ABAP-Variablen weicht oft von **lesbarer** Darstellung ab.

⇒ **Ausgabeaufbereitung** von ABAP-Variablen mittels **page**-Objekt

Verfügbarkeit in Layout + allen Eventhandlern.

Methoden mit *identischen* Parametern :

write( ) : Fügt aufbereiteten String in HTML der BSP ein.

to_string( ) : Liefert aufbereiteten String als return-Wert.

```
<%  page→write( value = sy-datum ).  %>
<%= page→to_string( value = sy-datum )  %>
```

```
<% page→write (
      value = sy-uzeit              " type any.  Einziger Pflichtparameter.  Name der Variablen
      format = if_bsp_page~co_format_long      " type i .  Ausgabeformat der Variablen
      outputlength = …              " type i.   Maximale Ausgabelänge des Strings
      num_decimals = …              " type i.    Anzahl darzustellender Nachkommastellen.
      reference_value = …           " type c.   Referenzwert - z.B. Währung oder Mengeneinheit.
   ).
%>
```

Es kommt auf die Art des value-Feldes an, ob alle Angabe Sinn machen und wirksam sind ...

- Jede BSP-Seite wird intern als Objekt ***page*** der Klasse **CL_BSP_PAGE** verwaltet, die das Inteface **IF_BSP_PAGE** implementiert. Diese Klasse ist Basisklasse aller BSPs. Ihre Methoden erlauben den Zugriff auf alle relvanten Informationen einer BSP.
- Im Grunde muss für alle Datums-, Zeit-, Währungs- und andere Mengenfelder vor der Seitenausgabe eine **Druckaufbereitung** (**Darstellungskonvertierung**) vorgenommen werden, um eine ansprechende, benutzer-freundliche Darstellung zu erhalten. Die interne Speicherung von ABAP-Variablen unterscheidet sich oft von der gewünschten Darstellung. So sieht man z.B. bei Ausgabe von <%=sy-datum%> für das **Datum** 20.03.2016 die Darstellung 20160320. Das gleiche Problem existiert bei **Zeit**- und **Währungsfeldern**.
- Die Methoden **write()** und **to_string()** berücksichtigen auch die im Benutzerprofil des Users hinterlegten persönlichen Einstellungen zur Formatierung. (TA **SU01** - Einstellung u.a. des Datums- und Währungsformats.)
- Wenn man ABAP-Variablen von Anfang an korrekt deklariert (z.B. Währungsdatentyp für eine Variable, die für einen Geldbetrag steht usw.), so ist es ausreichend, nur den Parameter value zu versorgen - weitere Formatierungsangaben sind nicht notwendig.
- Die **Formatierungsoptionen** erlauben die Aufbereitung von Datum, Zeit, Text, Zahlen und Währungen. Es muss sich um unstrukturierte Datentypen handeln. Folgende Integer-Konstanten sind im Interface **IF_BSP_PAGE** als Formatierungsoptionen vordefiniert:

  co_format_currency : Ausgabe im Währungsformat

  co_format_long : Lange Ausgabe co_format_short : Kurze Ausgabe

  co_format_lower : Ausgabe Kleinbuchstaben co_format_upper : Ausgabe Großbuchst.

  co_format_none : Default. Standardausgabe ohne Formatierung

ICF-Objekte in BSP-Anwendungen

Für laufende BSP-Applikation werden bestimmte **ICF-Objekte *automatisch* (implizit) instanziiert.**

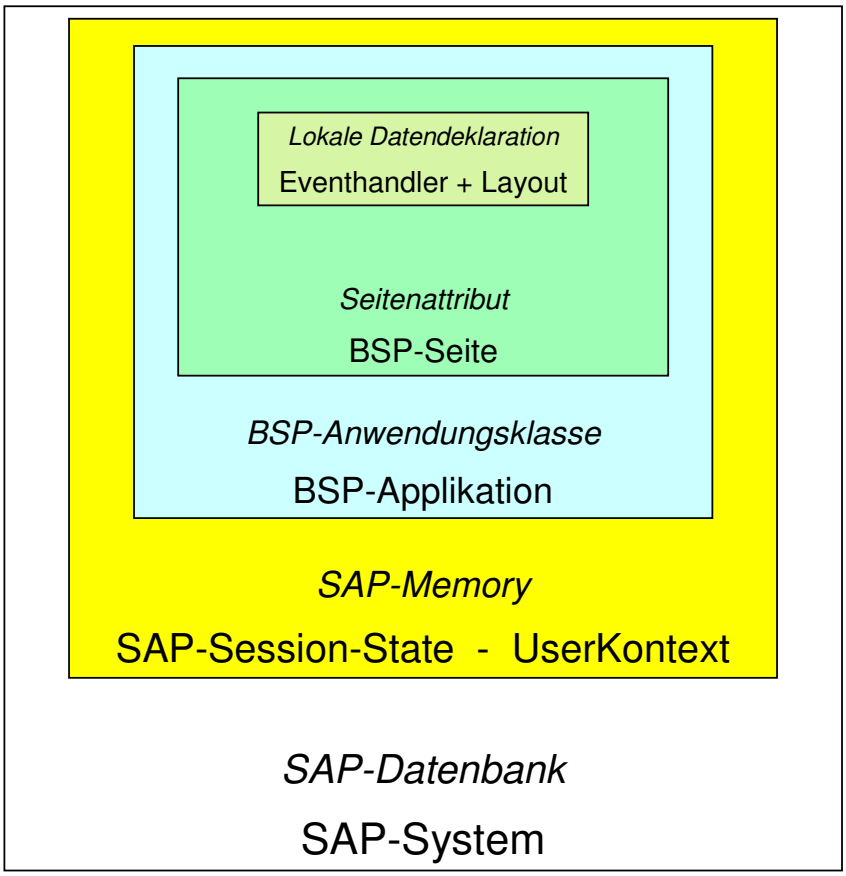
Stehen unter festen Namen zur Vfg. – allerdings nur zu bestimmten Zeitpunkten :

Objekt	Typ	Events	Methoden + Attribute (*nur kleine Auswahl* !)
navigation	if_bsp_navigation	2, 3, 4	set_parameter() goto_page() next_page()
event_id	String	4	
application	if_bsp_application	*immer*	
page	if_bsp_page	*immer*	write() to_string() get_page_name() get_page_url()
request	if_http_entity	2, 3, 4, 5	get_cookie() get_cookies() get_data()
runtime	if_bsp_runtime	*immer*	keep_context server session_id get_url() page_name()
response	if_http_entity	3, 5	set_cookie() set_data()

1	2	3	4	5
OnCreate	OnRequest	OnInitialization	OnInputProcessing	OnManipulation

- Je tiefer man in die BSP-Programmierung eindringt, desto öfter wird man auf automatisch instanziierte Objekte zugreifen. Dabei handelt es sich um Objekte entsprechender **ICF-Klassen** und **-Interfaces**, die eine Fülle nützlicher Methoden und Attribute zur Verfügung stellen.
- Auch wenn man das "high-level" BSP-Framework verwendet, wird man auf die **"low-level" ICF-Bestandteile** zurückgreifen, die dem BSP-Framework zugrunde liegen.

Stufen der Datenhaltung + Persistenz



← Impliziert auch verschiedene **Gültigkeitsbereiche** von Datendeklarationen.

Wichtig :

Gültigkeits**bereich**

Verwendbarkeit der Variablen ohne Neudeklaration.

≠

Gültigkeits**dauer**

Bewahrung + Zugreifbarkeit des **Variablenwerts**.

↓

In der Regel **verschieden**,

je nachdem ob Anwendung **stateful** oder **stateless** läuft.

- Im **Seitenlayout** oder den **Eventhandlern** deklarierte lokale Variablen können *nur dort* verwendet werden.
- **Seitenattribute** werden als Attribute der gesamten BSP-Seite abgelegt und können daher sowohl im Seitenlayout als auch allen Eventhandlern *dieser* Seite benutzt werden.
- Methoden und Attribute können auch für alle Seiten einer BSP-Applikation deklariert werden. Dies erfolgt in der (optionalen) **BSP-Anwendungsklasse** für alle Seiten einer BSP-Anwendung.
- Systemglobale Daten sind durch die BSP-Anwendung durch DB-Zugriffe zu verwalten und werden auf der **SAP-Datenbank** dauerhaft (sitzungsübergreifend) gespeichert.