A Developer's Guide to Creating Powerful and Flexible Web Applications with the New Web Application Builder

Karl Kessler

Karl Kessler joined SAP AG in 1992 as a member of the Basis modeling group, where he gained experience with SAP's Basis technology. In 1994, he joined the product management group of the ABAP Development Workbench. Since 1997, Karl has been Product Manager for SAP's business programming languages and frameworks.

(complete bio appears on page 58)

In Release 6.10, SAP retires the SAP Basis system and supplants it with the SAP Web Application Server (Web AS). The SAP Web AS provides the technical foundation for R/3 Enterprise,[1] and is also the translation technology for most of the other new mySAP.com offerings. The difference between the SAP Web AS and the SAP Basis system is that the Web AS includes *native* support for Internet protocols such as HTTP, which eliminates the need to set up a separate Internet gateway such as the Internet Transaction Server (ITS),[2] and also includes a brand-new development model called Business Server Pages (BSPs). BSPs enable you to build web applications that can dynamically access SAP data.

A BSP application is made up of a series of HTML pages that contain server-side scripts, complete with basic ABAP statements such as `SELECT`, and BAPI calls for accessing data on SAP components such as R/3, BW, etc. You can code these scripts with either ABAP or JavaScript. The choice is yours.

Whichever route you take — ABAP or JavaScript — you simply surround your code with standard tags, and the code is executed with a URL request. You can use any kind of HTML (Hypertext Markup Language) or WML (Wireless Markup Language) tags in your pages.

How does a developer actually build a BSP application? You build it with the Web Application Builder. Since the Web Application Builder is based on the ABAP Workbench, ABAP developers should

[1] R/3 Enterprise is the successor to R/3 Release 4.6C.

[2] That's good news for administrators!

find it easy to navigate about and use this tool. You'll be building new web applications and web UIs in the same way you've been building your ABAP applications. Transaction SE80 provides access to the structural overview of a BSP application. The ABAP Editor and Debugger are at your disposal. But the Web Application Builder is not limited to the ABAP development environment. This tool complies with the WebDAV[3] standard, which means you can use external editors (and/or employ skilled web designers who are adept at using those external editors) to design sophisticated user interfaces.

This article will acquaint you with the process. Here, I will guide you step by step through the development of a simple, two-page BSP application. By the end of this discussion, I think it will be clear that this new 6.10 tool uses an outside-in approach similar to the one you find on leading Java and Microsoft application servers.

Introducing Business Server Pages (BSPs)

Web applications based on the Business Server Pages principle consist of a sequence of HTML pages. The pages may contain any kind of standard HTML or WML tags. To dynamically include data from SAP and non-SAP systems, logic (expressed as script statements) is embedded within the page and processed upon request to produce the final, static HTML or WML page. In this way, data from SAP and non-SAP sources can be retrieved, processed, and formatted for output. This scheme parallels the approach taken by Java Server Pages (JSPs) and Active Server Pages (ASPs), which are widely used in web application development.

In the code sample below, you see a simple BSP in which the string `Hello World` is rendered five times in different font sizes. The static HTML text contains server-side ABAP code — in this case, a `do` loop that is executed to produce the web page dynamically. Inside the loop, the string is output. The font size, which is declared as an ABAP variable in line 2, is increased in line 7 before the loop is reentered:

```
1 <%@page language = "ABAP" %>
2 <% data: fontsize type I value 1. %>
3      <% do 5 times. %>
4      <font size = <%= fontsize %>
5            Hello World.
6      </font>
7      <% add 1 to fontsize. %>
8      <% enddo. %>
```

As I mentioned earlier, the script code of BSPs can be either JavaScript or standard ABAP.

> ✓ ***Tip***
>
> *The use of JavaScript is just a matter of syntactical taste. Both in ABAP and in JavaScript, the underlying data structures are ABAP variables such as fields, structures, or internal tables. The integrated JavaScript engine allows you to treat these variables as if they were JavaScript variables.*

During the lifetime of a BSP, script code is executed in response to various events. For example, an Initialization event would initiate the retrieval of data from database tables or the invocation of function modules — i.e., a PBO (Process Before Output) module. An Input Processing event would initiate a reaction to a submission of form data after user input — i.e., a PAI (Process After Input) module.

[3] Web-based Distributed Authoring and Versioning (**www.webdav.org**).

So how do you build a BSP application? You build it with the Web Application Builder.

The Web Application Builder

The Web Application Builder for BSP applications is a collection of integrated, individual tools that enable the implementation of complex web applications. Those who are familiar with ABAP Workbench-based development for the ITS runtime platform will find many similarities between the use of ITS and Web Application Builder tools. But do not be misled — BSP applications are not a substitute for ITS-based scenarios.

Figure 1 provides an overview of the Web Application Builder toolset. The main development tool is the BSP Editor, which is used to develop dynamic

BSPs vs. IACs

Those familiar with Internet Transaction Server (ITS) programming might ask, "What are the differences between a BSP application and an Internet Application Component (IAC) written with Business HTML?" The greatest difference is in the architectural approach. ITS is an Internet gateway that enables SAP transactions to run in a web browser and allows developers to create an HTML layout via an ordinary SAP screen with minimal effort. Since an ordinary SAP screen is the starting point, developers remain inside the classical SAP dialog protocol. On the other hand, BSP applications do not require developers to think in terms of SAP dialog steps, allowing a more flexible approach to building web applications that is based on dynamic pages scripted on the server side.

Figure 1 *The Web Application Builder Toolset*

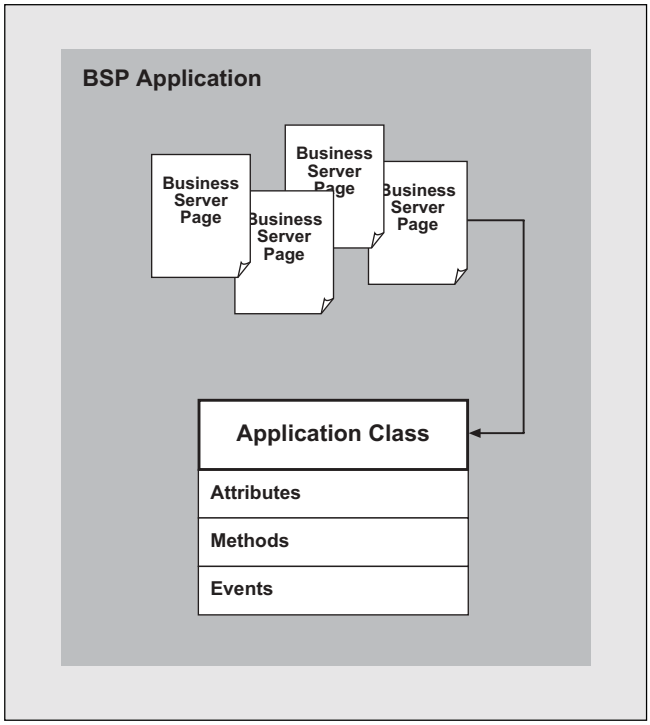
Tool	Features
BSP Editor	☑ Preview capabilities ☑ Basic script editing ☑ HTML code editing features (e.g., drag and drop) ☑ Syntax checks ☑ Support for third-party, WebDAV-compliant design and development tools (e.g., Adobe GoLive)
ABAP Debugger	☑ Integrated server-side ABAP and JavaScript debugging both in the BSP and the event-handling code
Tag Library	☑ Out-of-the-box libraries for standards like HTML, WML, and XHTML ☑ Drag-and-drop tag definitions and attributes
MIME Repository	☑ Stores images, icons, style sheets, etc. ☑ Accessible from within PC-based, WebDAV-enabled design tools ☑ Integrated with the Change and Transport System (CTS) for change control
Theme Editor	☑ Enables customer branding of SAP-standard Web Application Server applications ☑ Enables "default" images (and other MIME objects) to be overridden selectively

server pages — both the layout of a dynamic page and its associated event handlers. The Tag Library, MIME Repository, and Theme Editor are tools that ease the typical development tasks associated with HTML-based applications. These tools are explained in more detail later on in this article.

The main runtime tool is the well-known ABAP Debugger, which debugs BSP applications step by step. If you have developed the script code in JavaScript, a script debugger automatically pops up. This is a major difference compared to ITS development, where server-side debugging can become difficult. Since the Web Application Builder is integrated in the traditional ABAP Workbench environment, you can easily navigate to other Workbench tools, such as the ABAP Dictionary, the ABAP function and class libraries, and the ABAP Editor.

Figure 2 shows the anatomy of a BSP application. A BSP application consists of a couple of dynamic, scripted web pages, each of which has its own parameters to make the BSPs more flexible. Event handlers are attached on the page level. If you want to separate the BSP from its business logic to ease maintenance of the BSP, you can create an application class that contains all the methods that constitute the business logic.

Figure 2 Anatomy of a Web Application



> ✓ ***Tip***
>
> *A BSP is compiled into a local ABAP class. While this fact is an implementation detail, you can see it easily when debugging a BSP application. The page layout and the event handlers are methods of the generated class, and the script code embedded in the page is translated into code of the generated layout method. Likewise, the page parameters are also translated into method arguments of the generated class. During debugging, all of the generated method names become visible. This should help you understand the syntactical rules for page layout and event handlers.*

To demonstrate these concepts, let's look at the tasks involved in building a very simple, two-page web application.

Building a BSP Application: Creating the First Page

There are six basic steps involved in building a BSP application:

1. Define a new BSP application.
2. Add parameters to the page.
3. Mark up your page with HTML.
4. Embed data placeholders and code within the page.
5. Add event handlers.
6. Add the finishing touches.

Let's take a closer look at each of the steps by building a simple demo application.[4] This application

[4] If you want to try this yourself, you will need access to a 6.10 server, and you must have SAP GUI 6.10. You can order a 6.10 development system via the SAP Service Marketplace at **http://service.sap.com**.

Web AS vs. Internet Transaction Server

For many years, the Internet Transaction Server (ITS) has served as SAP's strategic platform for web application development. In addition to allowing customers (and SAP!) to build simplified, web-based R/3 transactions (known as Easy Web Transactions), ITS enables instant access to nearly all standard R/3 transactions and reports via its unrivaled SAP GUI for HTML component. For this reason, ITS is the platform on which all current SAP Internet solutions (e.g., ESS, BBP, CRM, Workplace) are built.

A key design goal of ITS from its very beginning was to allow web-enablement of almost any SAP transaction or report without significant reprogramming. Thus, by necessity, ITS was tailored to the traditional programming model of R/3, and excels in situations where dialogs to traditional R/3 transactions are required. On the other hand, for purely web-based applications to be developed from scratch, the more flexible BSP approach of the Web AS excels and is preferable as the more appropriate programming model that can easily address the technical challenges of stateless web protocols.

The Web AS and Java/J2EE

Having recognized the growing popularity and benefits of Java technologies, SAP has embraced Java as a strategic platform for web-enablement. SAP has joined several major consortiums and committees in an attempt to participate in the future direction of this powerful technology.

Internally, SAP has launched several serious projects utilizing Java, and has identified Borland's JBuilder as its IDE of choice. Externally, SAP's commitment to supporting Java is evidenced by the rich Java support planned for the upcoming 6.20 release of the Web AS. Boasting a fully functional, integrated J2EE server (InQMy), the Web AS promises developers groundbreaking options for building portable, scalable applications. Access to mySAP.com component systems is provided via the new Java Connector (JCo), now available for free from SAP at **http://service.sap.com/connectors**.

With the Web AS's Java support, developers can build powerful applications integrating data from mySAP.com component systems, SQL databases, and legacy systems. The range of options available with the Web AS brings developers closer than ever to the elusive vision of "any language, any development tool, any runtime platform."

Future releases of the Web AS will allow integration between Business Server Pages and Java components. Data will be freely available, and it will be possible to perform processing across heterogeneous components, regardless of whether they run on the ABAP, JavaScript, or Java technology stack. Developers will be able to choose the technology most appropriate for the task at hand and their experience.

> ✓ ***Tip***
>
> *It is perhaps little known that since 4.6C you can connect to external SQL databases from ABAP using the EXEC SQL statements in ABAP. You only have to issue an EXEC SQL statement with a suitable CONNECT string.*

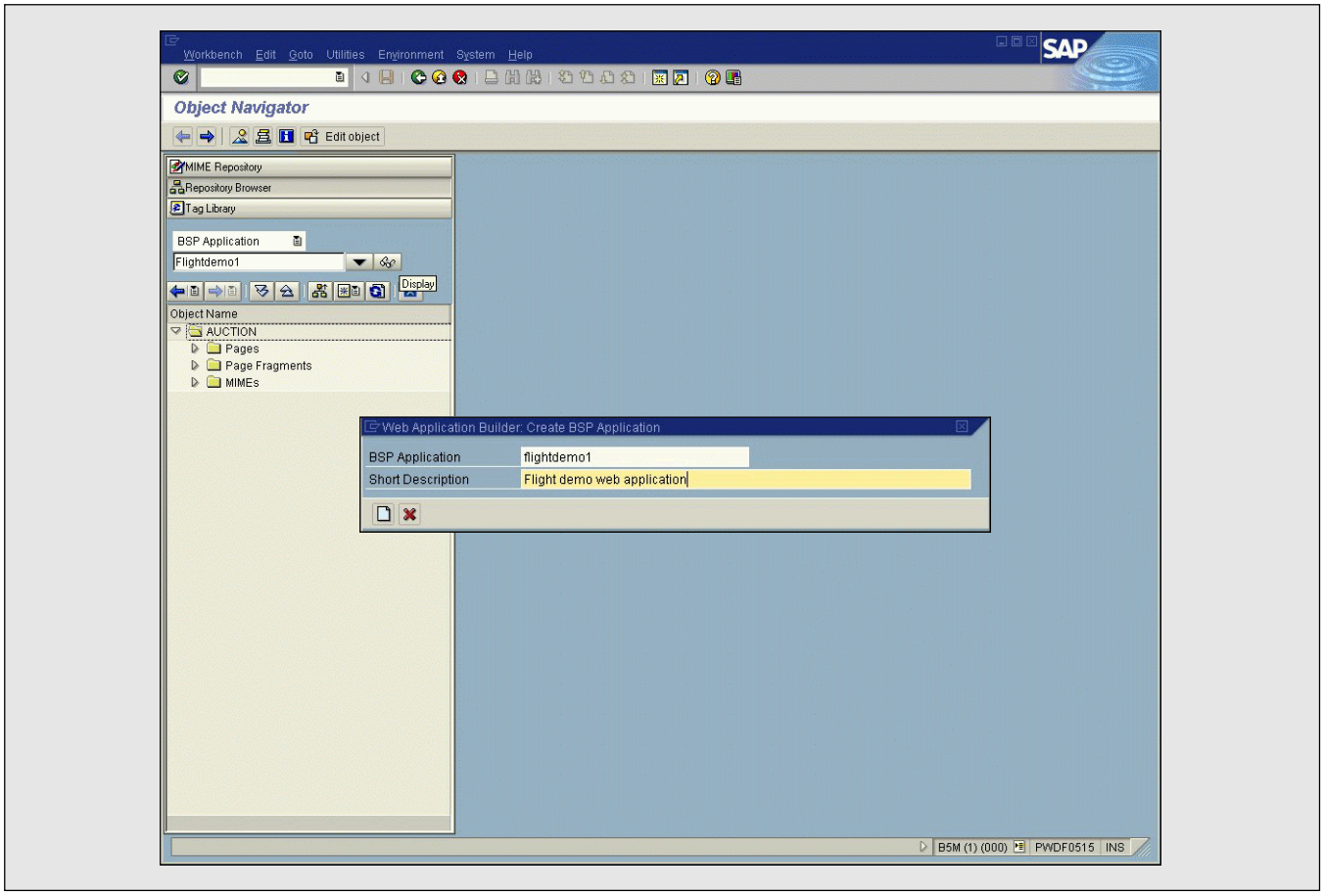
So how do you select a platform that best serves your needs? The table on the next page provides a comparison between the features of ITS and the Web AS to help you tailor your solution to your own unique environment.

(continued on next page)

(continued from previous page)

Feature	Internet Transaction Server	SAP Web Application Server (Web AS)	
		Business Server Pages	Integrated J2EE Server
Platforms	Windows/Linux	Windows/Linux	Windows/Linux
Programming languages	ABAP, HTML Business	ABAP, JavaScript	Java
Program model	Traditional ABAP with HTML templates	ABAP scripting	Servlet/JSP
Session management	Stateful (IACs); stateless (WebRFC/ Flow Logic)	Almost stateless; stateful is supported	Stateless
Programming style	Inside-out (IACs); outside-in (MiniApps)	Outside-in	Outside-in
SAP screen rendering	Built-in HTML GUI	No	No
Customer branding	Themes, CSS (style sheets)	Themes, CSS (style sheets)	CSS (style sheets)
Development tool	Web Studio or Web Application Builder	Web Application Builder and any WebDAV-compliant tool	JBuilder and add-ons
UI abstraction	HTML Business	Tag libraries (contain custom tags that offer more powerful elements and their own HTML renderers)	Tag libraries (contain custom tags that offer more powerful elements and their own HTML renderers)
Source management	File-based, repository	Repository	File-based
Interfacing with mySAP.com components	BAPIs, RFMs, SAP screens, reports	BAPIs, direct (if mySAP.com component is based on 6.10)	BAPIs
Syntax check	Compile time	Compile time	Compile time
Debugging	Supported	Supported	Supported
SQL database integration	No	Yes	Yes
Summary	**Strong inside-out access to mySAP.com; integrated WebGUI and web reporting**	**Strong outside-in platform for ABAP community; more flexible programming model**	**Strong outside-in platform for Java community; access to SQL databases via JDBC**

Figure 3 Select the Type and Name of the BSP Application and Provide a Description



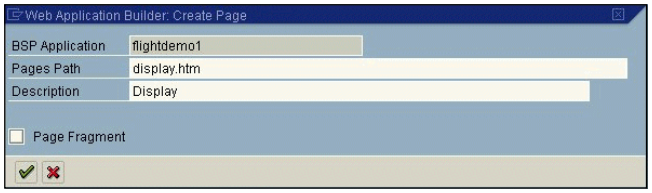
will display a list of flights from the well-known flight model used in SAP training courses. The records to be shown are contained in table SFLIGHT.

Step 1: Define a New BSP Application

To begin, we define a BSP application and a simple flight list display page as follows:

1. Log on to the SAP Web AS via the SAP GUI.
2. When the SAP Easy Access menu is displayed, launch transaction SE80 as you would with traditional development.
3. As shown in **Figure 3**, select "BSP Application" in the object type dropdown at the upper left of the screen and enter the name of your new BSP application ("Flightdemo1").
4. Press the "Display" button ( ). The system will warn you that the application doesn't exist and ask if you'd like to create it. Choose "Yes."
5. In the subsequent dialog, also shown in Figure 3, provide a description of your BSP application ("Flight demo web application") and press the "Enter" key. The system refreshes the left frame, displaying the root node "Flightdemo1" of your BSP application.
6. Double-click the "Flightdemo1" node in the left navigation frame. The properties of your BSP application will appear to the right.
7. To create the first page of your BSP application, right-click on the "Flightdemo1" node and select **Create → Page**.

Figure 4 Name the Business Server Page



8. As shown in **Figure 4**, a pop-up asks you for the name of your BSP. Type in "display.htm" and provide a suitable page description for the page, such as "Display."
9. The system opens the page in the editor frame of the "Layout" tab with some default HTML code (see **Figure 5**).

Step 2: Add Parameters to the Page

You have created your first Business Server Page, but it does not yet render, or output, any HTML on the screen. We could now start to add script code, but keep in mind that dynamic web pages may be reused later on, so to make the page as flexible as possible, let's first add parameters to the page. Page parameters can serve multiple purposes. If you want to share data in the page layout and the corresponding event handlers, or pass data from one page to another, page parameters are essential. Think of them like "method parameters." The page parameters become parameters of the underlying method of the generated local ABAP class. Due to the stateless nature of BSP applications, the parameter values are always

Figure 5 Default HTML Code for the Business Server Page

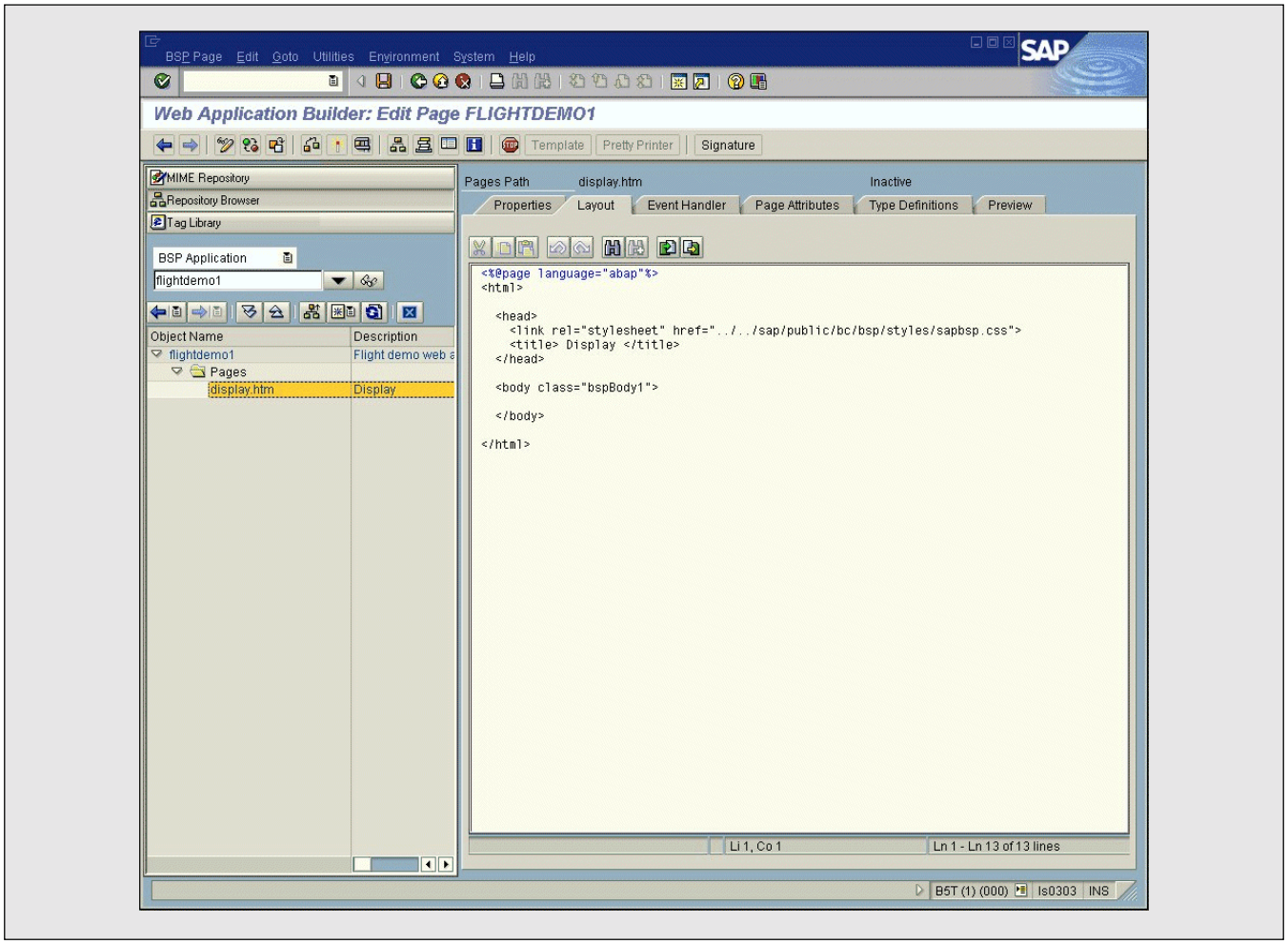
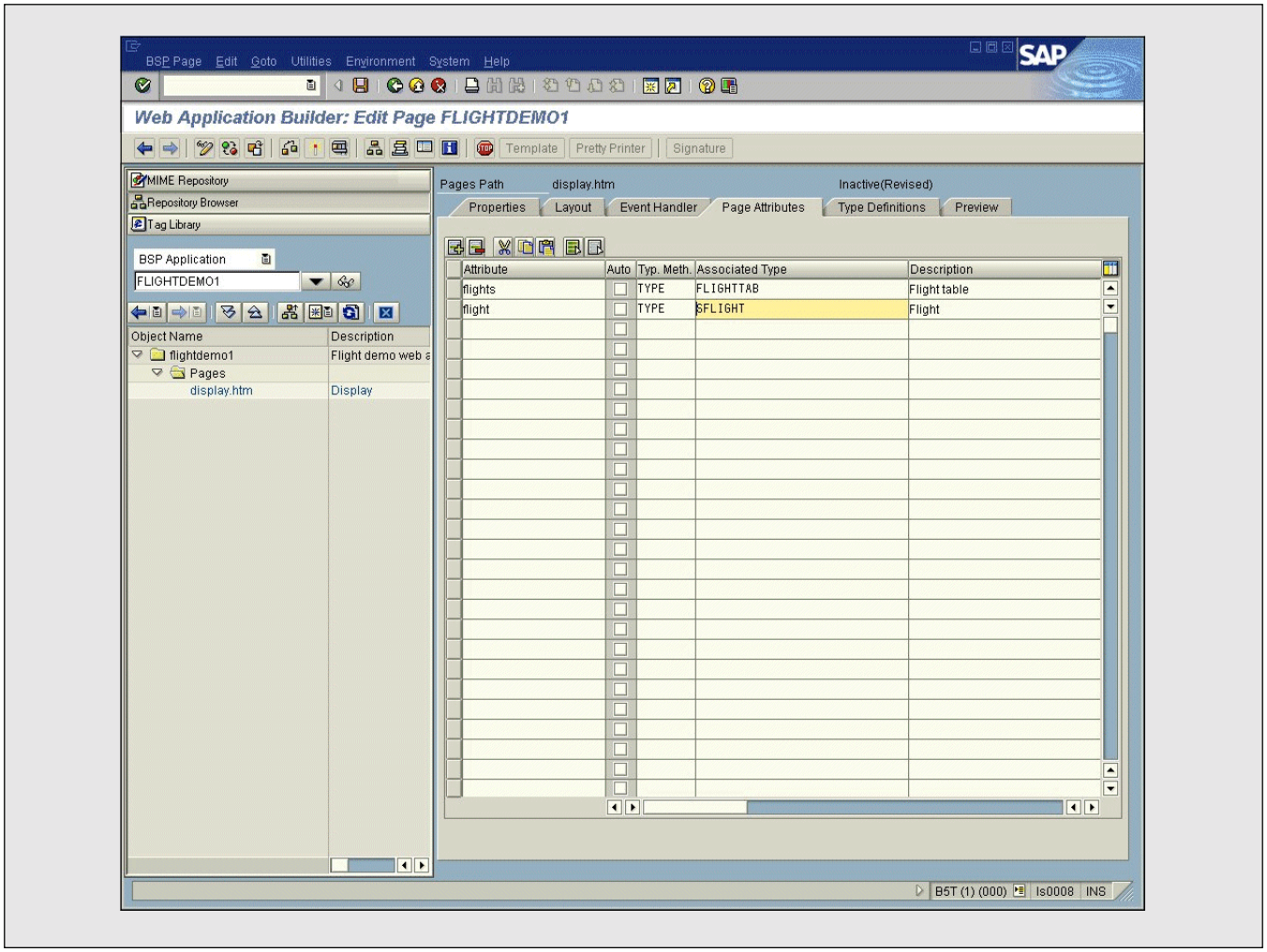


Figure 6 Parameters for the Business Server Page



initialized if a page is requested. Therefore, you cannot use the page parameters to keep a session context.

Add parameters to your "display.htm" page as follows:

1. Select the "Page Attributes" tab in the editor frame.

2. Type in a new parameter named "flights." You must provide an ABAP data type for this parameter, so enter "FLIGHTTAB" as the data type. As with the standard ABAP Editor, the data type field is linked to the ABAP Dictionary — double-clicking on this field causes the system to navigate to the ABAP Dictionary. You can see in the dictionary that "FLIGHTTAB" is simply a table of records having the structure "SFLIGHT." Thus the "FLIGHTTAB" parameter will be used to store a list of flights retrieved from table "SFLIGHT" at runtime.

3. Next, add a second parameter named "flight." You need this second parameter to iterate over the records of your internal table. Each record will be rendered dynamically. Enter "SFLIGHT" as the data type name for this parameter.

4. When finished, your entries should match those shown in **Figure 6**.

Step 3: Mark Up Your Page with HTML

Next, we are ready to provide the layout scripting code. Since our flights will be contained in the internal table FLIGHTS, we need to loop through it and render the HTML record by record. We use standard HTML tags for rendering.

If, for example, we want to display three columns of the FLIGHTS table, we need the following HTML skeleton:

```
<table>
    <tr>
        <td></td>
        <td></td>
        <td></td>
    </tr>
</table>
```

Coding this and other HTML constructs by hand requires a lot of tedious typing and a mental library of HTML tags and attributes. Fortunately, the Web Application Builder includes a Tag Library browser to help. The Tag Library browser, shown in **Figure 7**, enables easy insertion of HTML tags through drag and drop.

The HTML code just inserted by hand could have been inserted using the Tag Library browser as follows:

1. Open the Tag Library browser by pressing the corresponding toolbar button. This loads the browser into the navigation frame.

2. Expand the "HTML 3.2" branch to display the list of valid HTML tags.

3. Drag and drop the "<table>," "<td>," and "<tr>" tags into the editor frame of the "Layout" tab. The corresponding closing tags are inserted automatically.

4. To quickly code HTML fragments, highlight the "<td>" tag in the editor frame and copy it by holding down the control key.

Step 4: Embed Data Placeholders and Code Within the Page

Next, we use special scripting syntax to insert and format data at runtime. Close inspection of the first line within our auto-generated "display.htm" page reveals that the system suggests using ABAP for scripting:

```
<%@page language="abap"%>
```

Those of you who prefer to use JavaScript can express this preference by changing `abap` to `javascript` in the line of code above.

Lest you think the Tag Library browser is only useful for coding HTML tags, it also contains a library of script tags as well! Begin adding code to your page using the Tag Library browser as follows:

1. Open the Tag Library browser, if it is not already open, and expand the branch "BSP directives."

2. Drag and drop the code tag "<% … %>" just after the `<table>` tag in our page.

3. Highlight the code tag and copy it just before the closing `</table>` tag.

4. In the first code tag, type:

   ```
   Loop at flights into flight.
   ```

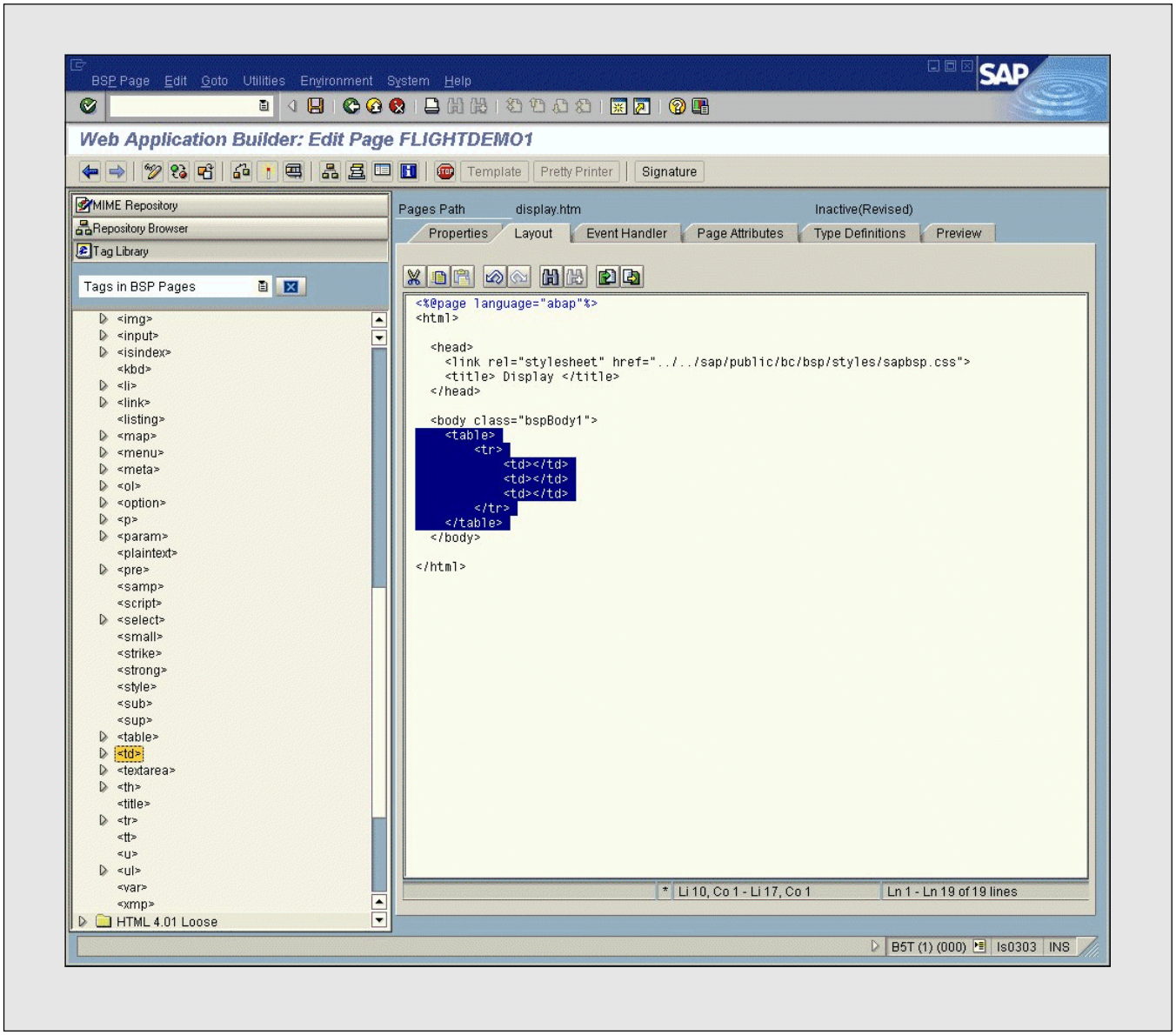
5. In the second code tag, type:

   ```
   Endloop.
   ```

6. Drag and drop the field evaluation tag "<%= %>" between the `<td>` and `</td>` tags.

7. Type in `flight-carrid`, `flight-connid`, and `flight-fldate` right after the equals sign (=).

Figure 7 *The Tag Library Browser*



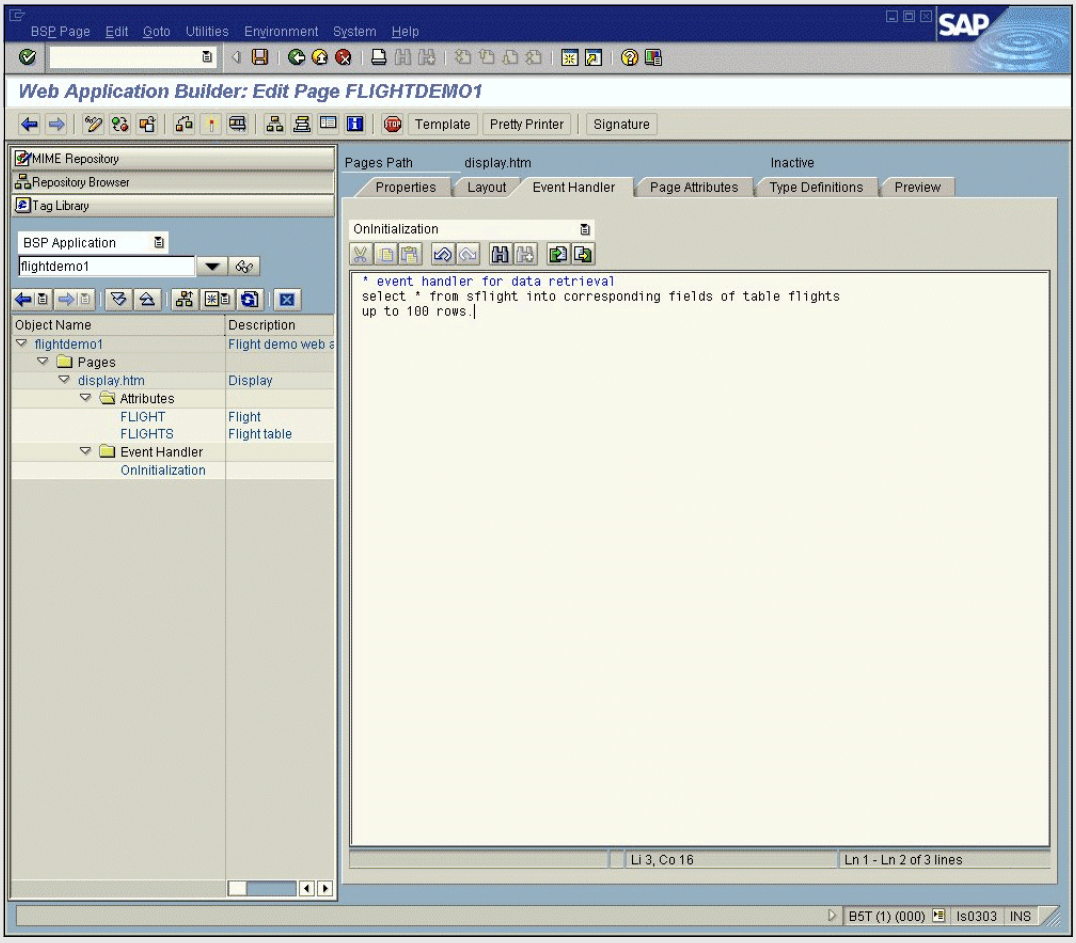
8. Save your work.

For those used to WYSIWYG editors such as Microsoft FrontPage, you might be thinking at this point that coding HTML this way is rather time-consuming. Remember that the Web Application Builder was designed to comply with WebDAV, a standard supported by many popular tools for the exchange of web application objects (e.g., HTML pages). Through WebDAV, pages designed in the editor of your choice (assuming it too is WebDAV-compliant) can easily be imported into the Web Application Builder. More on this later.

Step 5: Add Event Handlers

We have not yet implemented the code to retrieve flight information from table SFLIGHT. This will be

Figure 8 Implement Database Access with the Initialization Event Handler



done using the Initialization event handler, as illustrated in **Figure 8**:

1. Select the "Event Handler" tab.
2. Select the "OnInitialization" event handler in the dropdown listbox.
3. Add the following code:

```
select * from sflight into
  corresponding fields of table
  flights up to 100 rows.
```

4. Save your work.

Your page is now ready to be compiled:

1. Activate your page from the toolbar by pressing the button. The page will be tested for syntactic and semantic correctness, and a local ABAP class will be generated containing methods that correspond to your event handlers and layout specifications.
2. Select the "Properties" tab to see the URL that will load the page.
3. Press "F8" and the system loads the URL by launching the web browser.
4. A logon prompt appears, and after successfully logging in, our result page ("display.htm") is displayed. Obviously, the data displayed on your page will depend on the contents of table

Listing 1: Add Column Headers to the Business Server Page

```
<table>
    <th>Carrid</th>
    <th>Connid</th>
    <th>Fldate</th>
<% loop at flights into flight.%>
    <tr>
        <td><%=flight-carrid%></td>
        <td><%=flight-connid%></td>
        <td><%=flight-fldate%></td>
    </tr>
<% endloop.%>
</table>
```

SFLIGHT in your system. Use the Data Browser (transaction SE16) to add rows if data is missing.

Step 6: Add the Finishing Touches

So what is missing from your first page? For readability, you should add column headers. Again, you can use the Tag Library browser or simply type in the corresponding `<th> </th>` tags and supply appropriate column names. Your code should now look like **Listing 1**, and your result page like **Figure 9**.

Congratulations! You've created your first BSP application!

Advanced Formatting

While functional, the user interface for our demo application is simplistic. Real-world pages tend to have many more visual elements, often in an attempt to "brand" the site. Standard HTML offers a rich library of tag sets and attributes with which to implement a compelling design, but hard-coding formatting options leads to an application that cannot be maintained. Especially for large applications, it would be ideal to have some way to centrally store site design elements (e.g., button and background colors, or type styles) so that changes can be propagated instantly throughout the site.

Figure 9 The Resulting Display Page

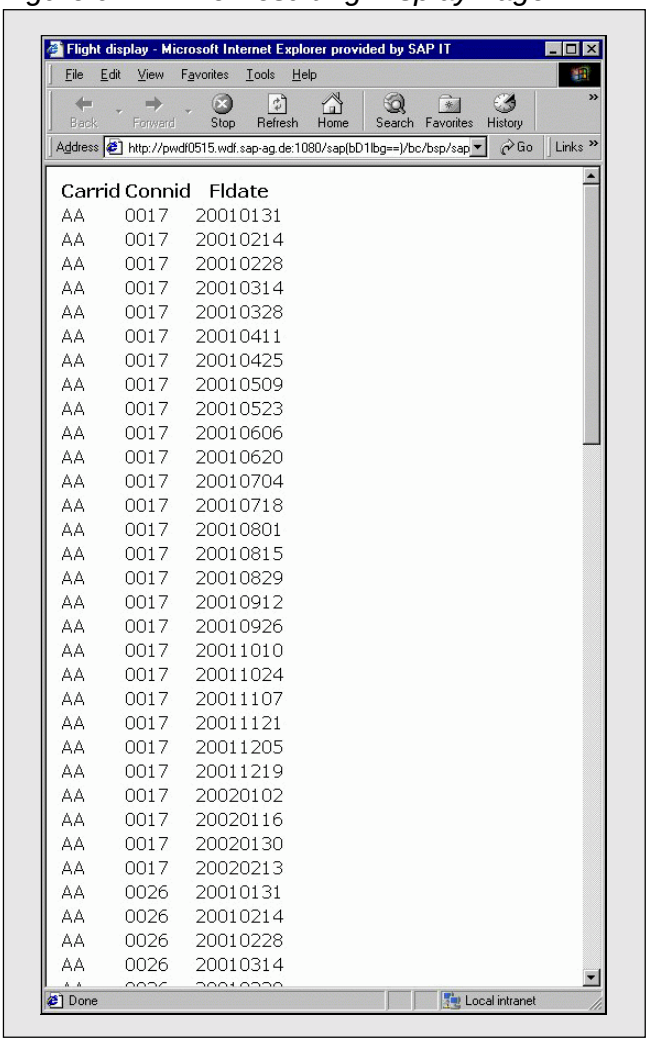
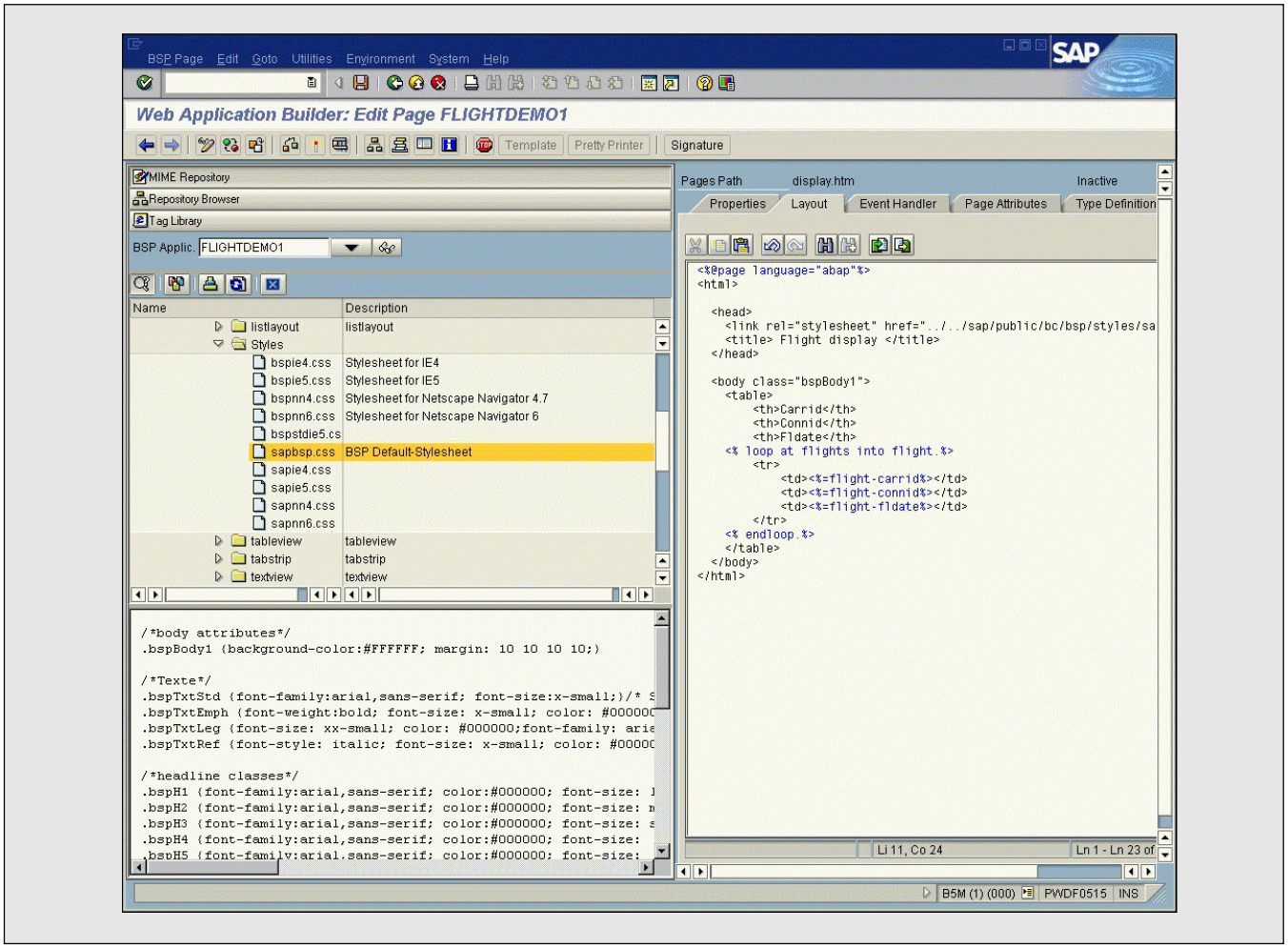


Figure 10 Expand the "Styles" Node to View the Style Sheets



Enter Cascading Style Sheets (CSS)! CSS allows you to define visual elements in a separate file known as a *style sheet*, and refer to that style sheet in any of your application's HTML code. Maybe you have already noticed a reference to a style sheet in the header of the initially generated HTML code (refer back to the code in Figure 7):

```
<link rel="stylesheet" href="../../
sap/public/bc/bsp/styles/sapbsp.css">
```

Whether created outside or inside the Web Application Builder, style sheets — and all MIME objects, such as images, for that matter — must be stored in something called the MIME Repository. The MIME Repository is simply a storage area reserved for binary and other objects to be used in web applications, such as images, sounds, and style sheets. Objects created outside of SAP can be uploaded to the repository. More on this later.

For now, let's improve our page's appearance using an existing style sheet from the repository:

1. Bring up the MIME Repository by pressing the corresponding button in the left navigation frame.

2. Two folders are displayed, both belonging to the SAP namespace: the "PUBLIC" folder and your BSP application's folder ("FLIGHTDEMO1"). Expand the "PUBLIC" branch, and then expand all nodes that correspond to the path name of

Listing 2: Modified HTML Layout Code for the Display Page

```
<%@page language="abap"%>
<html>

  <head>
    <link rel="stylesheet" href="../../sap/public/bc/bsp/styles/sapbsp.css">
    <title> Flight display </title>
  </head>

  <body class="bspBody1">
    <table class="bspTbvStd">
        <th class="bspTbvHdrStd">Carrid</th>
        <th class="bspTbvHdrStd">Connid</th>
        <th class="bspTbvHdrStd">Fldate</th>
    <% loop at flights into flight.%>
    <% data: str type char10.
       write flight-fldate to str. %>
        <tr>
            <td class="bspTbvCellStd"><%=flight-carrid%></td>
            <td class="bspTbvCellStd"><%=flight-connid%></td>
            <td class="bspTbvCellStd"><%=str%></td>
        </tr>
    <% endloop.%>
    </table>
  </body>
</html>
```

the style sheet reference in your HTML page's head, as noted in the previous line of code (i.e., "...public → bc → bsp → styles").

3. After expanding the "Styles" node, you will see a listing of style sheets (see **Figure 10**). If you double-click on a style sheet's name (e.g., "sapbsp.css"), an editor frame opens in the lower left containing the style sheet's class definitions.

4. Resize the editor frame for readability and scroll down to the very end, where you will find style classes for table rendering.

5. Drag and drop the style names into the corresponding HTML tags in the editor frame of the "Layout" tab at the upper right. Precede each style name with the string `class=`, for example:

```
<table class="bspTbvStd">
```

6. While not directly relevant to our CSS example, let's take this opportunity to improve the formatting of our "flight date" field. Replace the `<%=flight-fldate%>` string in the template with the following:

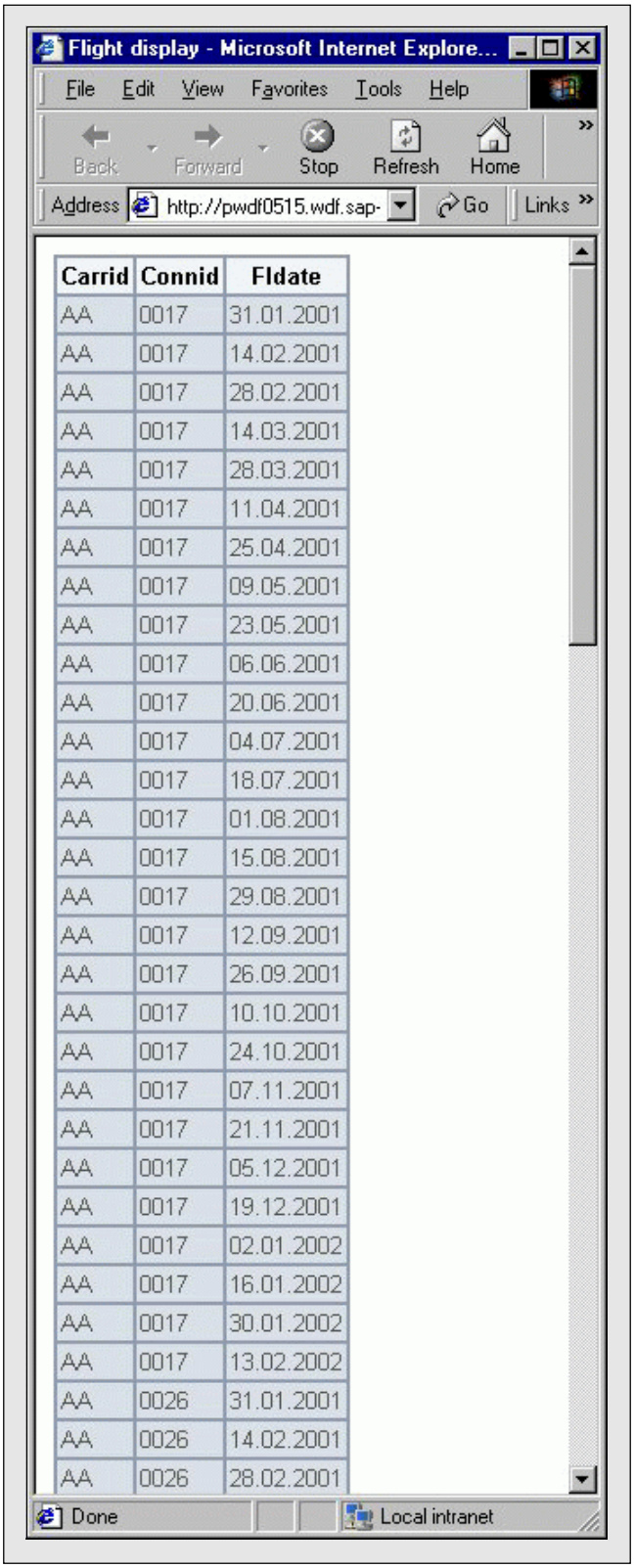
```
<% data: str type char10.
   write flight-fldate to str. %>
```

   This code converts the internal date format to the user's preference for date formatting according to the user's master record in the database.

7. Before continuing, ensure your code now matches that in **Listing 2**.

8. Activate your page and re-launch it. As you can see in **Figure 11**, the tabular output will look much better than it looked before (refer back to Figure 9).

Figure 11 The Improved Display Page



Carrid	Connid	Fldate
AA	0017	31.01.2001
AA	0017	14.02.2001
AA	0017	28.02.2001
AA	0017	14.03.2001
AA	0017	28.03.2001
AA	0017	11.04.2001
AA	0017	25.04.2001
AA	0017	09.05.2001
AA	0017	23.05.2001
AA	0017	06.06.2001
AA	0017	20.06.2001
AA	0017	04.07.2001
AA	0017	18.07.2001
AA	0017	01.08.2001
AA	0017	15.08.2001
AA	0017	29.08.2001
AA	0017	12.09.2001
AA	0017	26.09.2001
AA	0017	10.10.2001
AA	0017	24.10.2001
AA	0017	07.11.2001
AA	0017	21.11.2001
AA	0017	05.12.2001
AA	0017	19.12.2001
AA	0017	02.01.2002
AA	0017	16.01.2002
AA	0017	30.01.2002
AA	0017	13.02.2002
AA	0026	31.01.2001
AA	0026	14.02.2001
AA	0026	28.02.2001

Processing User Input: Creating the Second Page

So far, we have seen how to render dynamic web pages that include SAP data. But how do we handle user interactions such as submitting HTML forms?

To demonstrate, let's add a selection screen to our demo application upon which a user can enter a carrier ID to sort the resulting flights list. We'll create an initial page called "search.htm" for our web application and implement the OnInputProcessing event for that page:

1. Bring up the Repository Browser by pressing the corresponding button in the navigation frame.

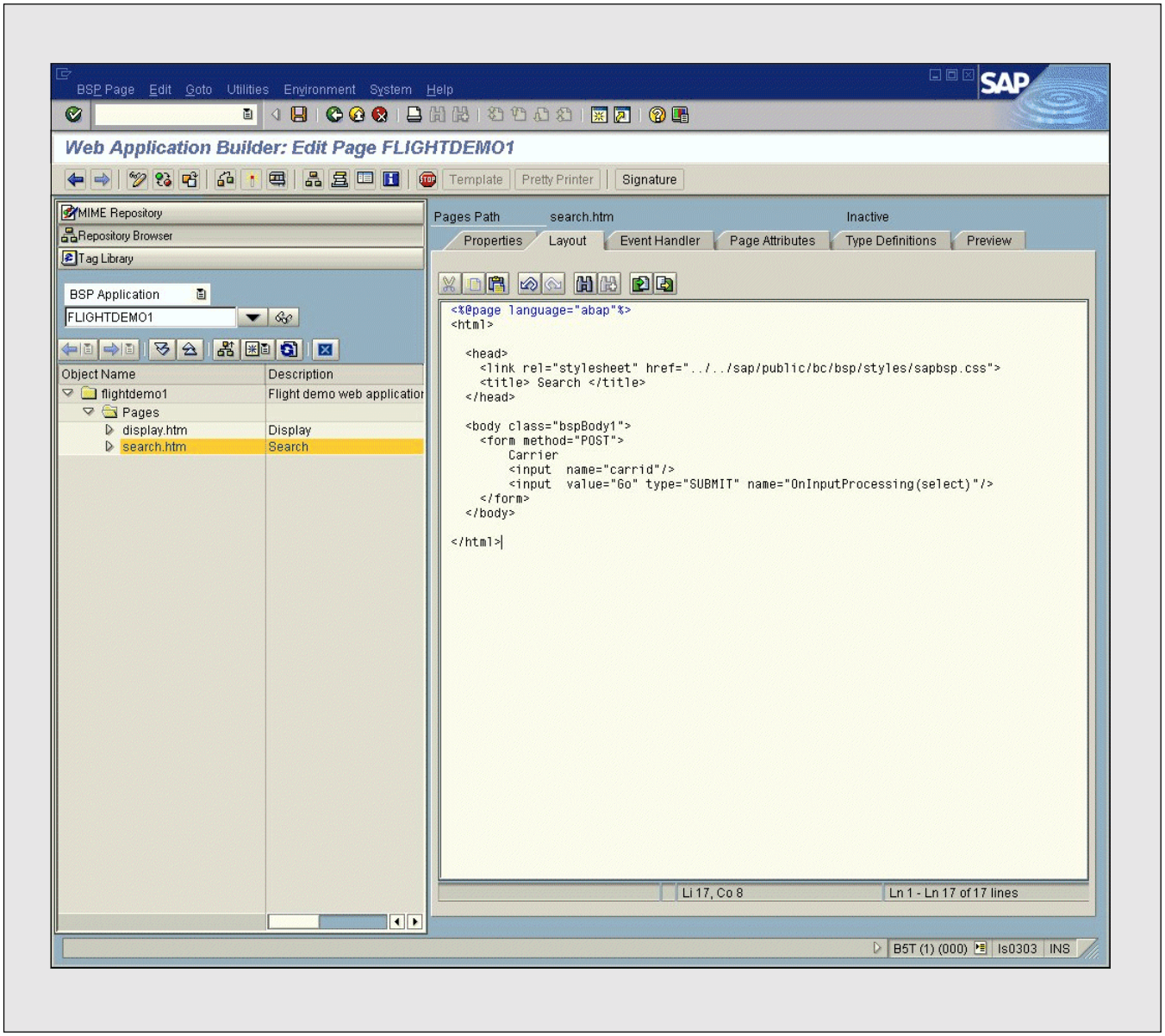
2. Open the context menu and add another page to your BSP application, following the steps described in the previous section.

3. Name your page "search.htm."

4. Insert the following code in the editor frame of the "Layout" tab, as shown in **Figure 12** (again, you can drag and drop HTML tags from the Tag Library browser if you wish):

```
<form method="POST">
    Carrier
    <input name="carrid"/>
    <input value="Go" type="SUBMIT"
      name="OnInputProcessing(select)"/>
</form>
```

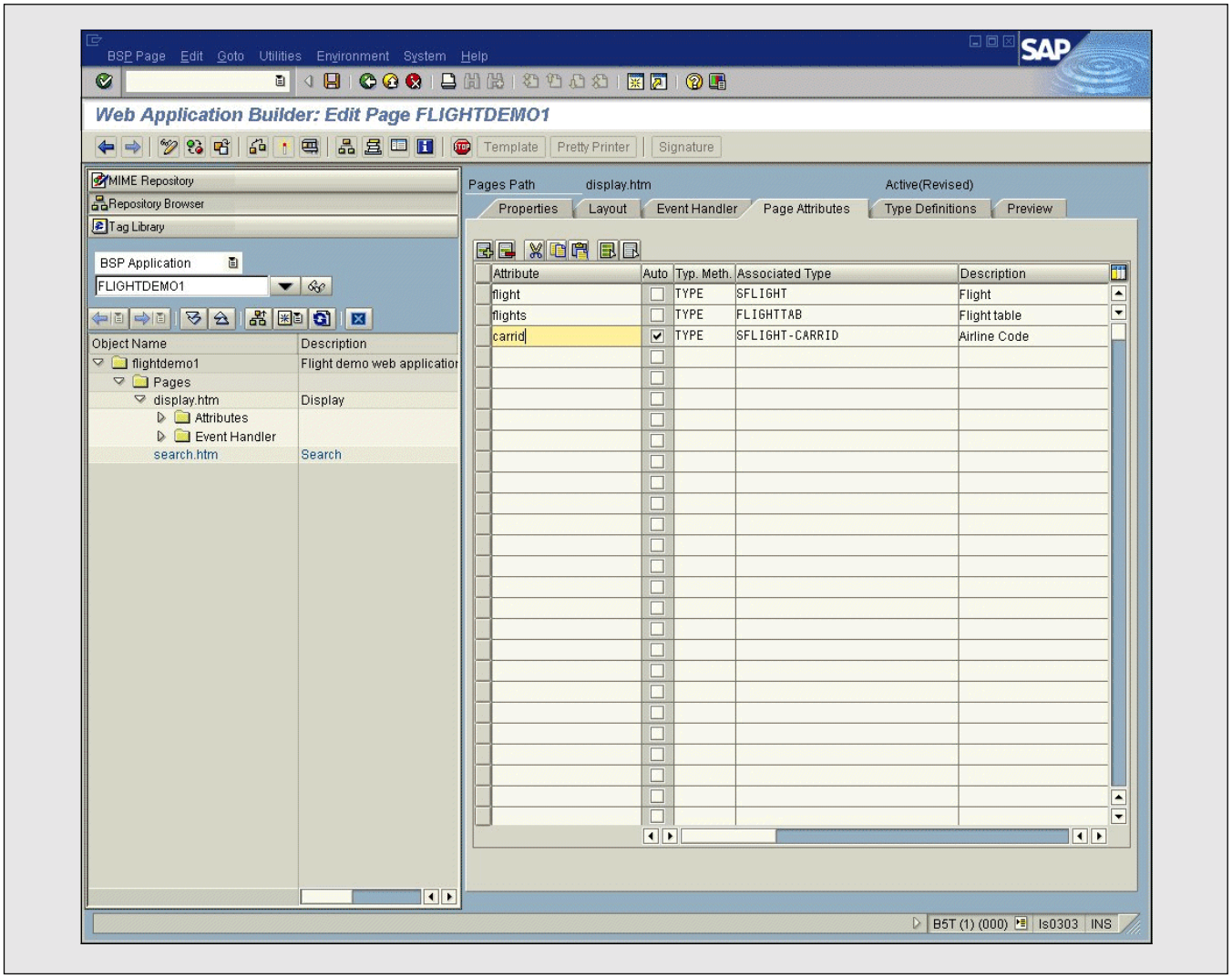

Figure 12 Add Input Fields to the Search Page



This code needs a bit of explanation. First, to submit fields to a server from a browser, you surround the fields with `<form> </form>` tags. Here, we have two input fields: one field serves as selection criterion for carriers preceded by the label "Carrier," and the other is a submit button with the label "Go." When the form is submitted by pressing "Go," control is passed to the Web AS, which invokes the page's OnInputProcessing event. Note that in pages with multiple buttons, distinct values can be substituted for the SELECT string, allowing the code in the event handler to determine which button was pressed.

Before implementing the OnInputProcessing event handler for this page, let's first modify our original "display.htm" page to support this new capability to sort by carrier ID. To do this, we must add a parameter to the "display.htm" page so that the carrier

Figure 13 Modify the Display Page to Support Sorting by Carrier ID

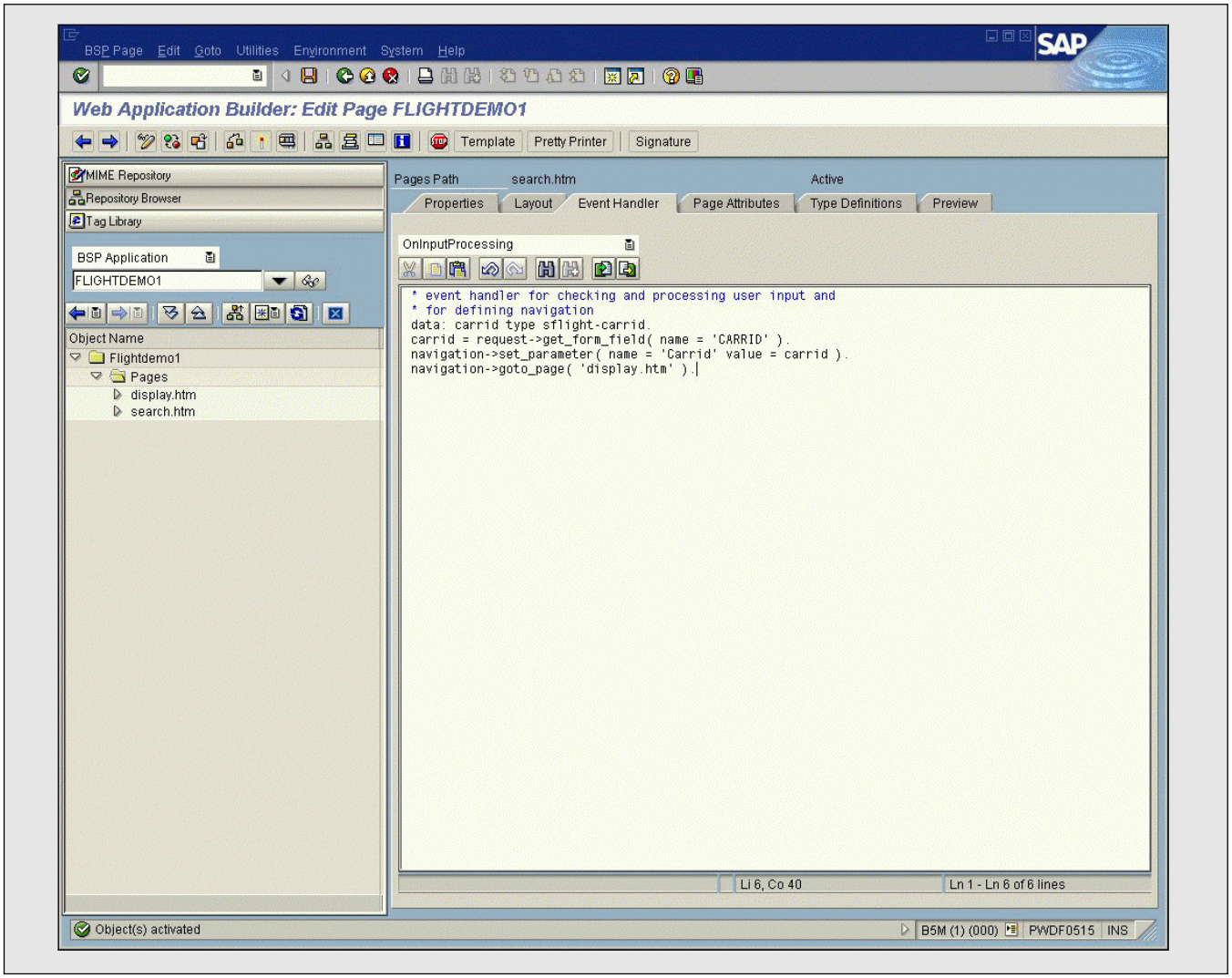


ID entered by the user will be passed to the page (see **Figure 13**):

1. Double-click the "display.htm" page in your navigation frame.
2. Select the "Page Attributes" tab.
3. Add a new parameter named "carrid."
4. Check the "Auto" flag. This ensures that when the page is requested the parameter is filled with a URL parameter that is set accordingly (i.e., it contains a corresponding `parameter = value` substring).
5. Enter "SFLIGHT-CARRID" as the ABAP data type name for the new parameter.
6. Select the "Event Handler" tab and choose the "OnInitialization" event.
7. Add `where carrid = carrid` to your `SELECT` statement (refer back to Figure 8), so that your code now reads as follows:

```
* event handler for data retrieval
select * from sflight into
  corresponding fields of table
  flights up to 100 rows
where carrid = carrid.
```


Figure 14 Add Parameter Passing and Page Navigation to the Search Page



> ✓ ***Tip***
>
> *If you want to test the page independently, press "F8." Since the parameter is not filled, an empty table will be shown. Add ?carrid=AA to the URL in your browser and reload it. Only flights from the AA airline will be shown.*

To complete our flight plan scenario, we have to program parameter passing and page navigation into the search page:

1. Double-click on the search page ("search.htm").
2. Select the "Event Handler" tab and choose "OnInputProcessing."
3. Enter the following code (see **Figure 14**):

```
data: carrid type sflight-carrid.
carrid = request->get_form_field
  ( name = 'CARRID' ).
navigation->set_parameter( name =
  'Carrid' value = carrid ).
navigation->goto_page
  ( 'display.htm' ).
```

This code retrieves the content of the form field "carrid" that is passed upon submission of the HTML form. Then the page parameter "carrid," which we recently added to the parameters list of our results page (see Figure 13), is set. Finally, via an HTTP redirect, the "goto_page" instructs the Web AS to begin processing "display.htm."

> ✓ ***Tip***
>
> *After examining the code in Figure 14, you might be wondering how the Request and Navigation objects come into play. Every event handler has some automatically generated parameters whose methods help to investigate the environment of a specific request or prepare a subsequent action such as navigation. Press the "Signature" button from the toolbar for a full list of predefined parameters, as shown below:*
>
> | Typ | Parameter | Type Assgn. |
> |---|---|---|
> | | RUNTIME | TYPE REF TO IF_BSP_RUNTIME |
> | | PAGE | TYPE REF TO IF_BSP_PAGE |
> | | REQUEST | TYPE REF TO IF_HTTP_REQUEST |
> | | NAVIGATION | TYPE REF TO IF_BSP_NAVIGATION |
> | | EVENT_ID | TYPE STRING |
>
> *If you double-click a parameter, you navigate directly into the Class Builder where you can inspect the public methods of that parameter.*

Now our demo application is fairly complete and can be run. **Figure 15** displays the results. Use the browser's "Back" button to work with different airline codes.

> ✓ ***Tip***
>
> *When you press "F8" to request and view your page, the URL of the page you are currently editing is automatically requested. You can define a different start page in the BSP application's "Property" tab by entering a different URL.*

Testing and Debugging

Since the Web Application Builder supports the same navigation, editing, and debugging features that the traditional ABAP Workbench supports, ABAP programmers will find it relatively easy to develop web applications for the Web AS. Creating pages, modifying their attributes, and adding code to event handlers is as easy as traditional ABAP programming. This is not surprising, as BSP applications are just another object type, like function groups, ABAP programs, or ABAP classes.

When developing ITS-based applications within the ABAP Workbench, you follow a similar procedure, but there is a major difference between ITS-based applications and BSP applications when it comes to the runtime environment. ITS resides on a separate server, so HTML templates designed for ITS need to be transferred from the development repository to this server. Since ITS is an HTML gateway, and therefore not based on the ABAP runtime system, ABAP programmers have less control over it using traditional ABAP-based tools. With BSP applications, on the other hand, the development and runtime environments are running in the same process family. You can carry out a syntax check of the ABAP Dictionary and repository at build-time, and then debug the session immediately as you would with a traditional ABAP program. Many of the ABAP productivity tools, such as runtime analysis, can be used to test your new web applications.

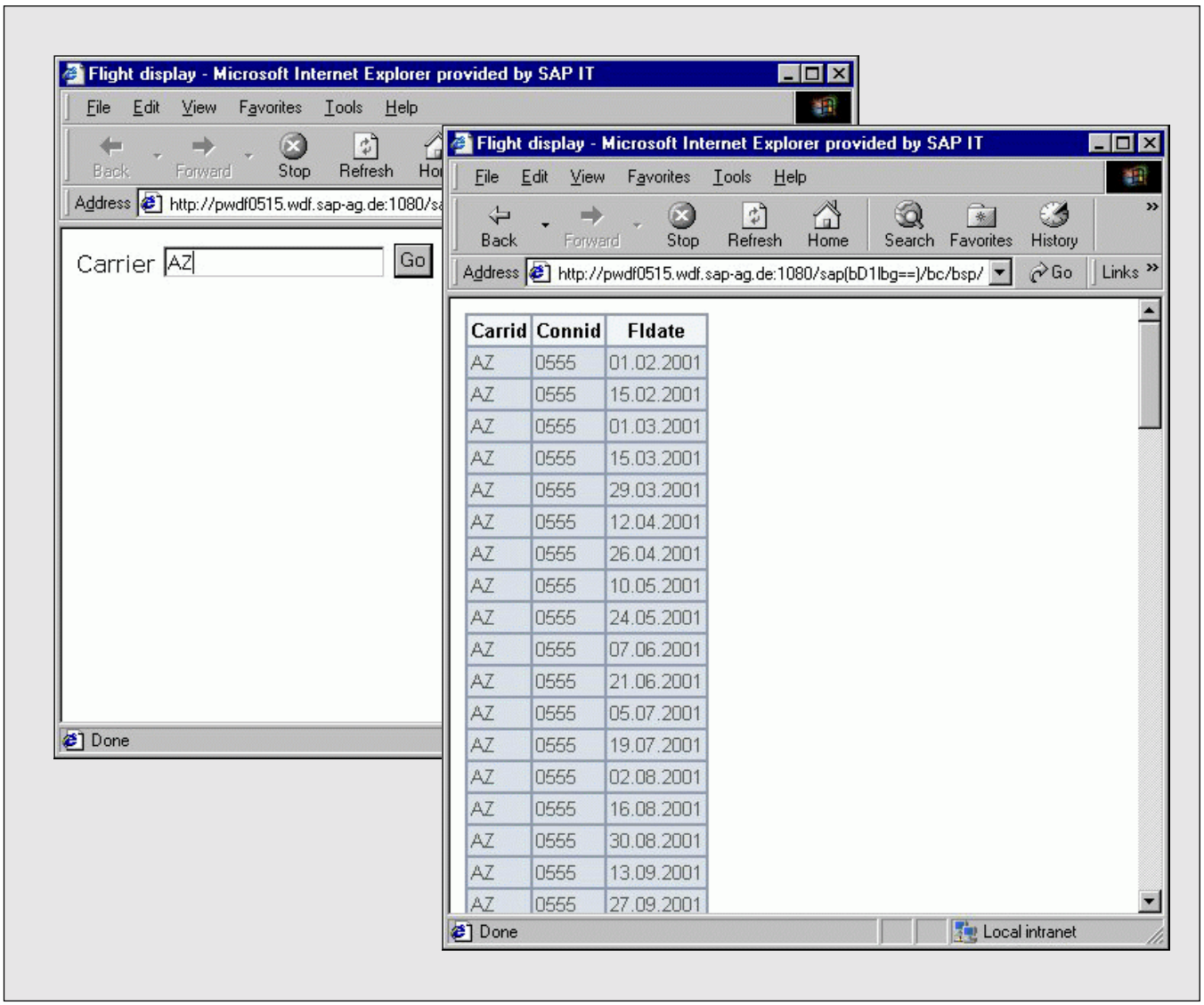
Debugging in the Web Application Builder is a snap:

1. Set breakpoints within one or more event handlers using the ABAP Editor.

2. Launch the application from a browser.

3. When a breakpoint is encountered, the Web AS automatically launches the ABAP Debugger in a separate SAP GUI window on the client PC. Here, you can single step through the code, examine the call stack, set watchpoint conditions, etc.

Figure 15 Search for and Display All Carriers with the ID "AZ"



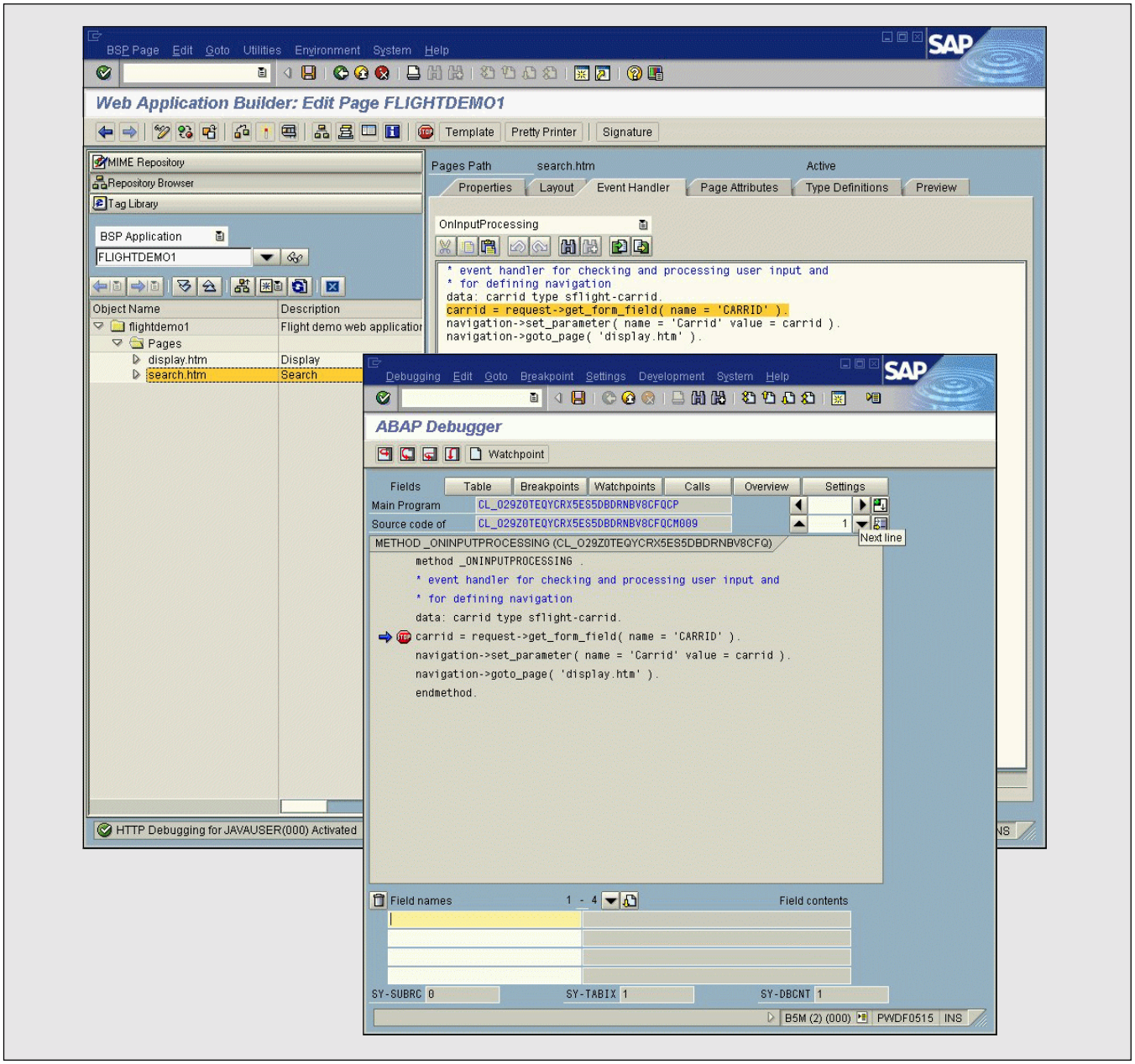
> ✓ ***Tip***
>
> *As with ITS, you can also debug a session in progress — every URL request can be debugged. Just set a breakpoint anywhere in the page (layout, event handler) and the ABAP Debugger will pop up when the page is requested (provided that the corresponding event handler is reached).*

The ease of debugging stems from the fact that, as in the traditional client/server world, the SAP Web AS is both a development and a runtime environment. Let's walk through the steps involved in debugging the "search.htm" page we created for our demo application:

1. Double-click the "search.htm" page.

2. Go to the "Event Handler" tab and select the event handler for OnInputProcessing.

Figure 16 Set a Breakpoint at Which the Debugger Will Stop



3. Set a breakpoint in the code, as shown in **Figure 16**.
4. Press “F8” to request and load the search page.
5. Type “AA” in the “Carrier” field and press “Go.”
6. The ABAP Debugger window pops up (also shown in Figure 16) and the debugger stops at your breakpoint.
7. Step through the code. You may have noticed that you are inside the generated method of the corresponding generated class of your BSP application.
8. Press the “Continue” button ( ) in the ABAP Debugger toolbar. The debugger pop-up will disappear and the browser displays the results page.

The MIME Repository and WebDAV Access

The MIME Repository easily stores images, icons, style sheets, etc., and its contents are also accessible from within PC-based, WebDAV-enabled design tools like Microsoft FrontPage. Let's take a closer look at these two functions of the Web AS and how they work together.

The MIME Repository

Perhaps the largest business benefit derived from web-based applications is ease of use. Proper design of an application's user interface is critical and usually requires a specialized skillset, so on many teams this responsibility is delegated to web designers and illustrators, rather than to programmers. Along with designing the user interface, these designers often create images, style sheets, and other related files with specialized tools. In addition, all of these items need to be deployed and version-controlled like the code developed by programmers.

Earlier we saw how the MIME Repository is used to store style sheets for inclusion in your BSPs. The MIME Repository can also meet the need for regulated deployment and version control by serving as a centralized object store. Images and styles built by designers with external tools can be easily uploaded and are immediately accessible to developers working on the application's logic. At runtime, the Web AS serves all images referenced in BSPs from the MIME Repository. In this way, the MIME Repository functions as a content management solution.

From a development perspective, the MIME Repository is divided into several sub-hierarchies: the SAP namespace, and a sub-hierarchy for every BSP application. The "PUBLIC" folder, which we used earlier when searching for styles, contains application-independent MIME objects. Within the sub-hierarchy for your BSP application, you can freely create subfolders as in a normal file system. Managing MIME objects and referring to them in your BSP application is easy.

> ✓ ***Tip***
>
> *For those who have developed web applications in the past, this approach differs from the traditional approach of deploying MIME files directly to a directory on the web server. Here, MIME objects are physically stored in the Web AS database, and are then served at runtime. To mitigate performance concerns, at runtime the Web AS makes image binaries available in a special Internet communication cache to reduce database I/O time.*

> ✓ ***Tip***
>
> *I would not recommend placing customer objects in the "PUBLIC" folder, as the customer content may be overwritten when upgrading to the next release. Instead, create a dummy public customer web application to contain these objects and refer to them with a UNIX-like syntax, for example: .../ZPUBLIC/IMAGES/LOGO.jpg.*

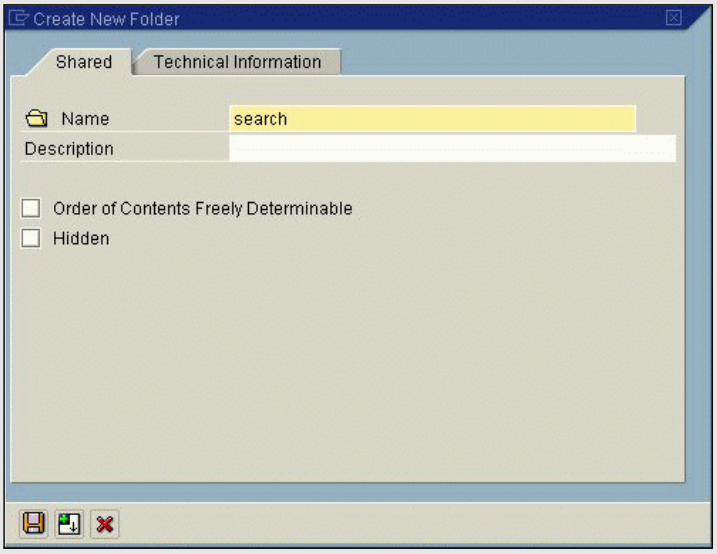
For some hands-on experience with the MIME Repository, try the following with our demo web application:

1. Bring up the MIME Repository by pressing the corresponding button in the navigation frame. Only the "PUBLIC" branch and your application-specific branch ("Flightdemo1") will be visible.

> ✓ ***Tip***
>
> *If your screen does not contain the "MIME Repository" button, you can add it. Navigate to **Environment → Settings**, and switch to the "Workbench (General)" tab. Select "MIME Repository" from the list of tools that appears.*

2. Right-click on your BSP application's name ("Flightdemo1") to open the context menu. Select "Create new folder."

Figure 17 Specify the Folder Name



3. As shown in **Figure 17**, specify a folder name (e.g., "search") and description and press "Save" ( ).

4. Right-click on your new folder and select **Import** → **MIME object**. A file selection box will appear.

5. Select an image file from the file system. Press the "Enter" key, and the image is uploaded into the MIME Repository and appears in the folder.

6. Double-click the image name for a short preview (see **Figure 18**). You can see here that I've selected the image file "sap_corporate_4c_tm.gif."

7. If not already open, open the "search.htm" Business Server Page we created for our demo application in the previous section and navigate to the editor frame of the "Layout" tab.

8. Bring up the Tag Library browser in the left frame by pressing the corresponding button.

9. Drag the "<img>" tag to the editor frame, below the `<body>` tag but above the `<form>` tag.

10. Expand the "<img>" folder and drag the "src" attribute into the image tag. Your finished image tag should look like this:

```
<img src="">
```

11. Finally, to insert the link to your image, bring up the MIME Repository again in the left frame and drag your image's name between the double quotes of the image tag's "src" attribute.

12. Activate and re-launch your page. As shown in **Figure 19**, your image is now displayed above the input field.

WebDAV Access

The contents of the MIME Repository constitute a virtual file system that is visible to WebDAV clients.

Figure 18 Preview of the Selected Image

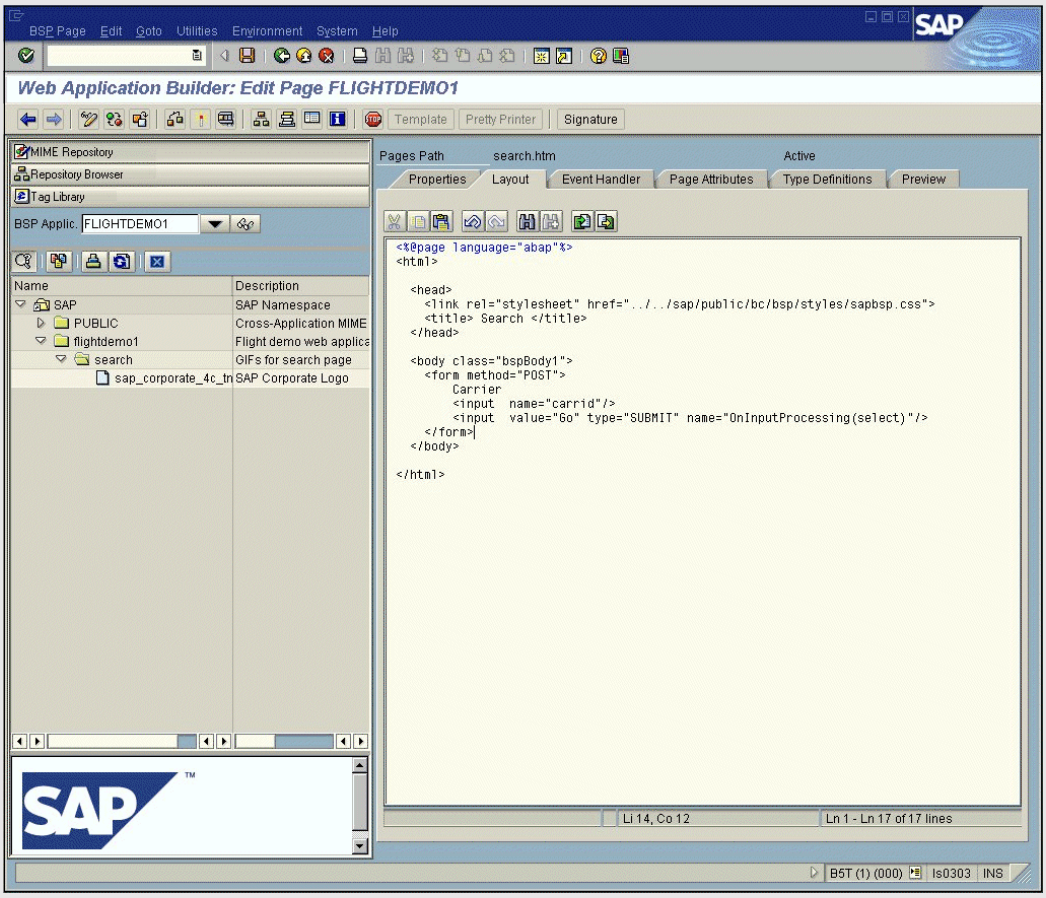


Figure 19 The Image Is Displayed on the Search Page

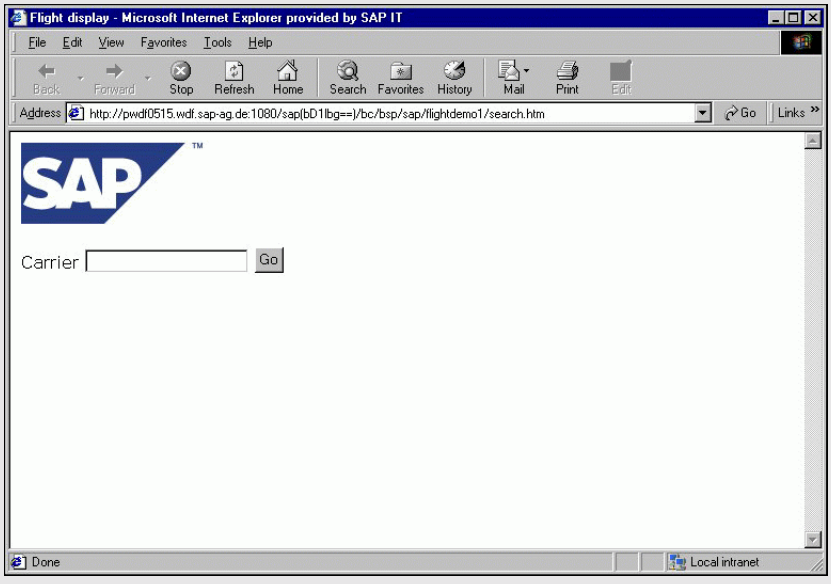
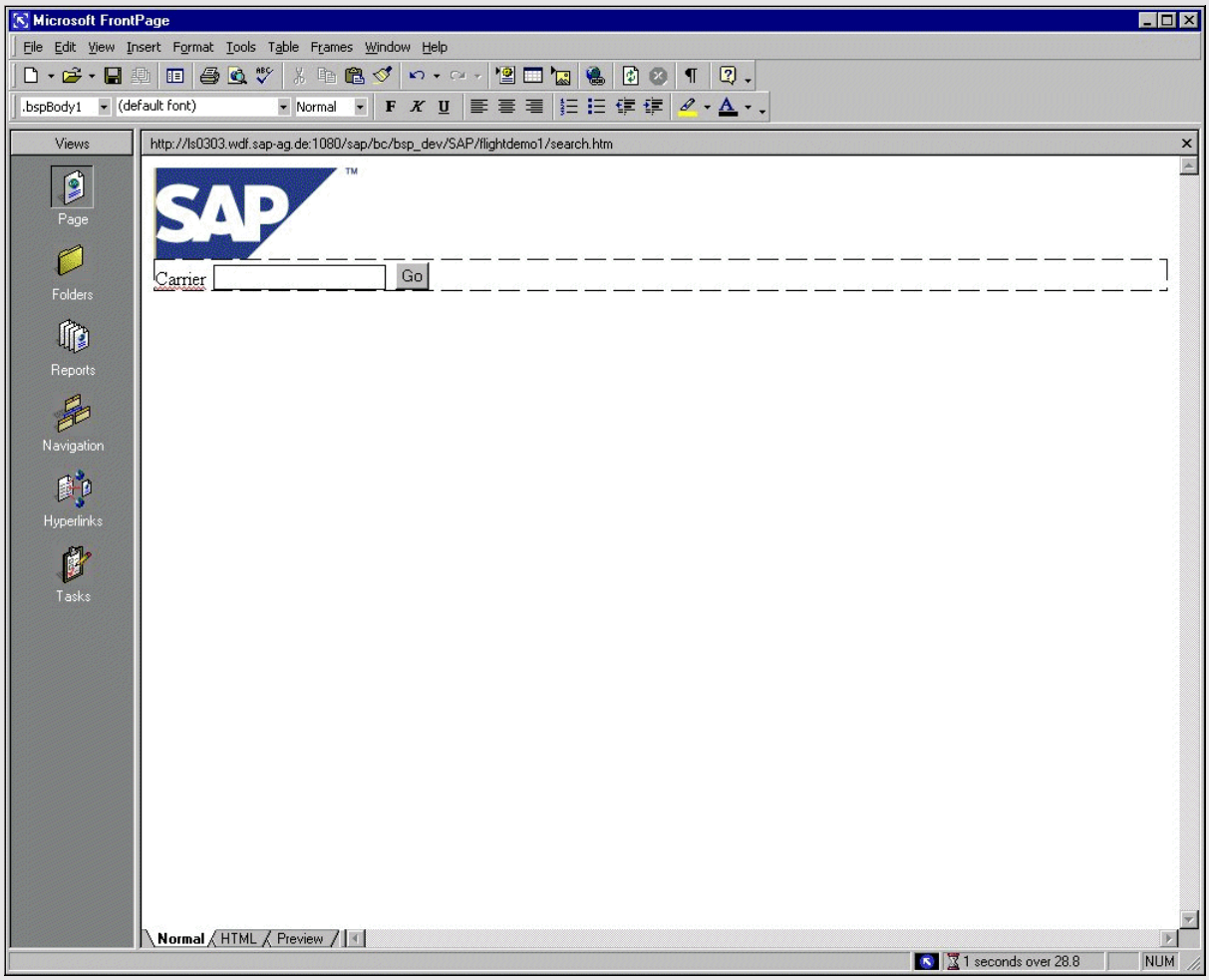


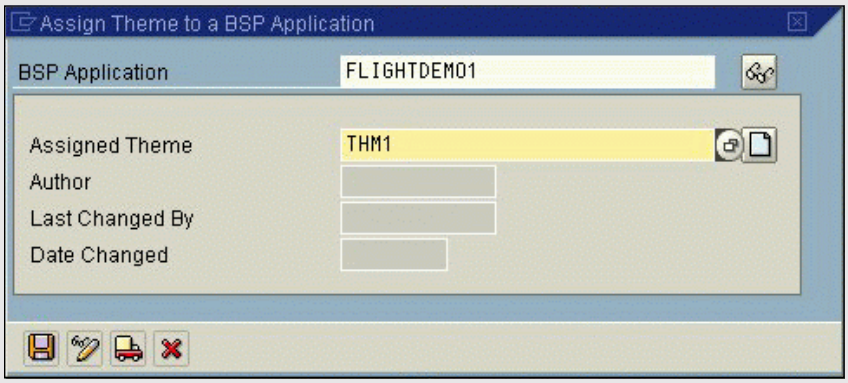
Figure 20 MIME Repository Files Are Accessible from Within Microsoft FrontPage



Windows NT Explorer is an example of a simple WebDAV client (others include Adobe GoLive, and Windows 98 and 2000). Let's take a look at this virtual file system using the Windows NT Explorer:

1. Launch the Windows NT Explorer from your desktop.
2. Add a new web folder using the wizard located in the folder "Web Folders." The location of the web folder is `http://<SAP Web AS>:<Port No>/sap/bc/bsp_dev`. (If you're using Windows 2000, first go to **Tools** → **Map Network Drive**, then click on the "Create a shortcut to a Web Folder" link.)
3. Browse the virtual file system as you would any other folder.
4. As demonstrated in **Figure 20**, you can open BSP projects and other MIME Repository files directly from within a WebDAV-enabled web design tool like Microsoft FrontPage. Any changes made using this or other WebDAV-enabled design tools instantly updates the images and HTML pages in the MIME Repository.

Figure 21 *Specify a Theme Name*



> ✓ ***Tip***
>
> *Keep in mind that WebDAV is only an interface to the MIME Repository, which means that page templates are treated as binaries in Microsoft FrontPage and need to be updated by a programmer in the BSP application.*

> ***Warning!***
>
> *Be careful to avoid overwriting objects of the "PUBLIC" MIME Repository folder, as all applications referring to the changed object will be affected.*

Theme Editor

In many situations, you'll wish to customize the appearance of existing SAP standard web applications or customer web applications, or you may want to support multiple appearances (e.g., for different corporate divisions) for a given application. Themes provide a simple way to accomplish this task.

Themes are simply collections of MIME Repository objects representing different appearances of an application. When a BSP application refers to a theme, the Web AS subsequently serves MIME objects stored in the repository under that theme. Thus customer images and styles can be used with existing BSP applications simply by adding a theme.[5]

Themes are currently restricted to images, icons, and styles only. A theme concept for BSPs is in development, which would enable customization of page layouts as well.

In practice, working with themes is easy. For example, try creating a theme as follows:

1. Double-click your BSP application's name ("Flightdemo1").

2. Press the "Assign Theme" button on the application toolbar.

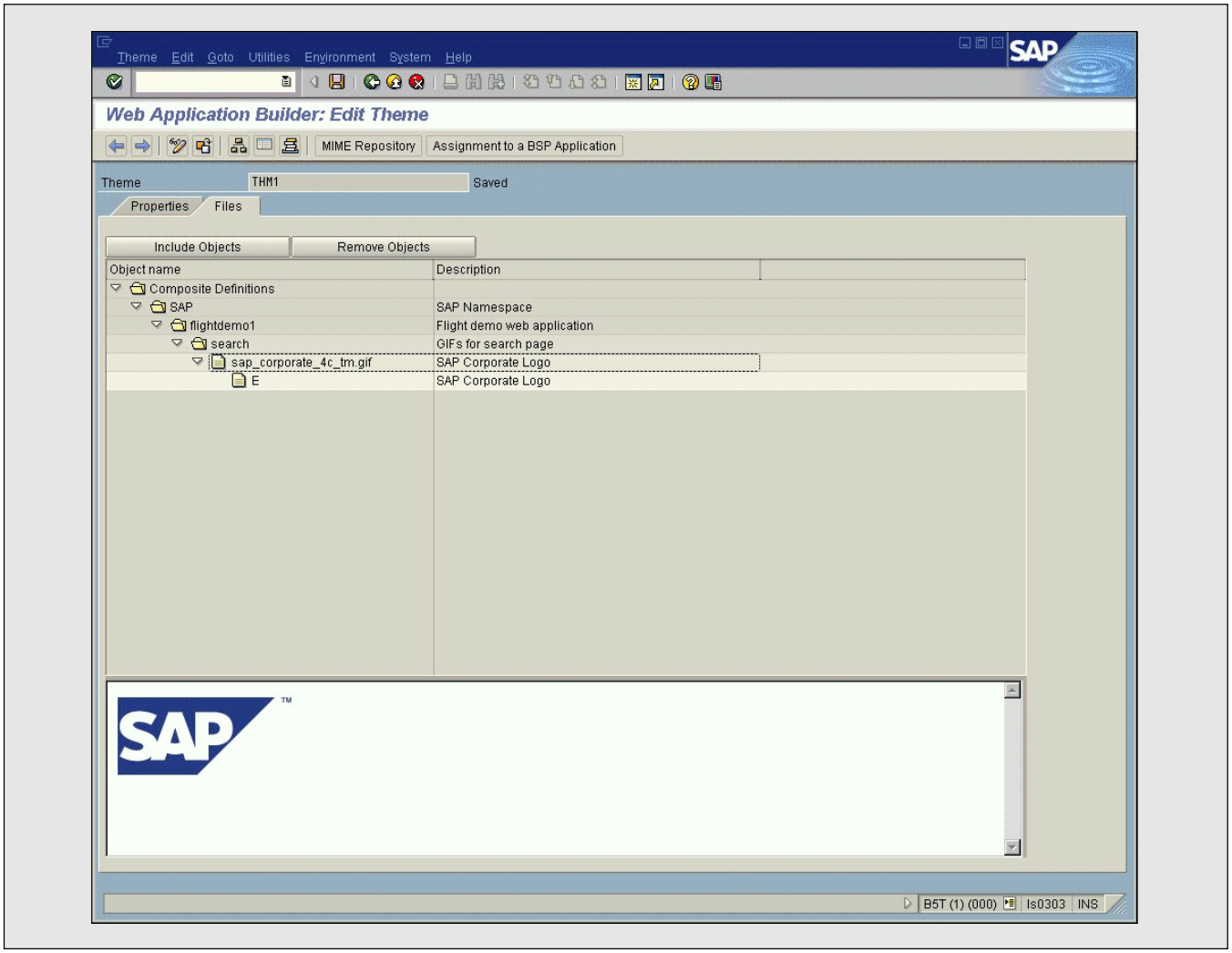
3. As shown in **Figure 21**, specify a theme name ("THM1") and press "Save" ( ).

4. In the theme editor, select the "Files" tab and press the "Include Objects" button.

5. A pop-up displays the MIME Repository. Select the MIME Repository object you would like to copy and customize. Upon adding this first

[5] A customer-based theme approach and corresponding customizing procedures are in development for the 6.20 release of the Web AS.

Figure 22 Adding a MIME Repository Object to the Theme



object, the theme's hierarchy of included objects is updated, as shown in **Figure 22**. Here, I've added the image file "sap_corporate_4c_tm.gif."

6. Right-click on the newly added image and select "Import" from the pop-up menu. The file selection dialog appears, enabling you to upload a new graphic or MIME object and add it to the theme.

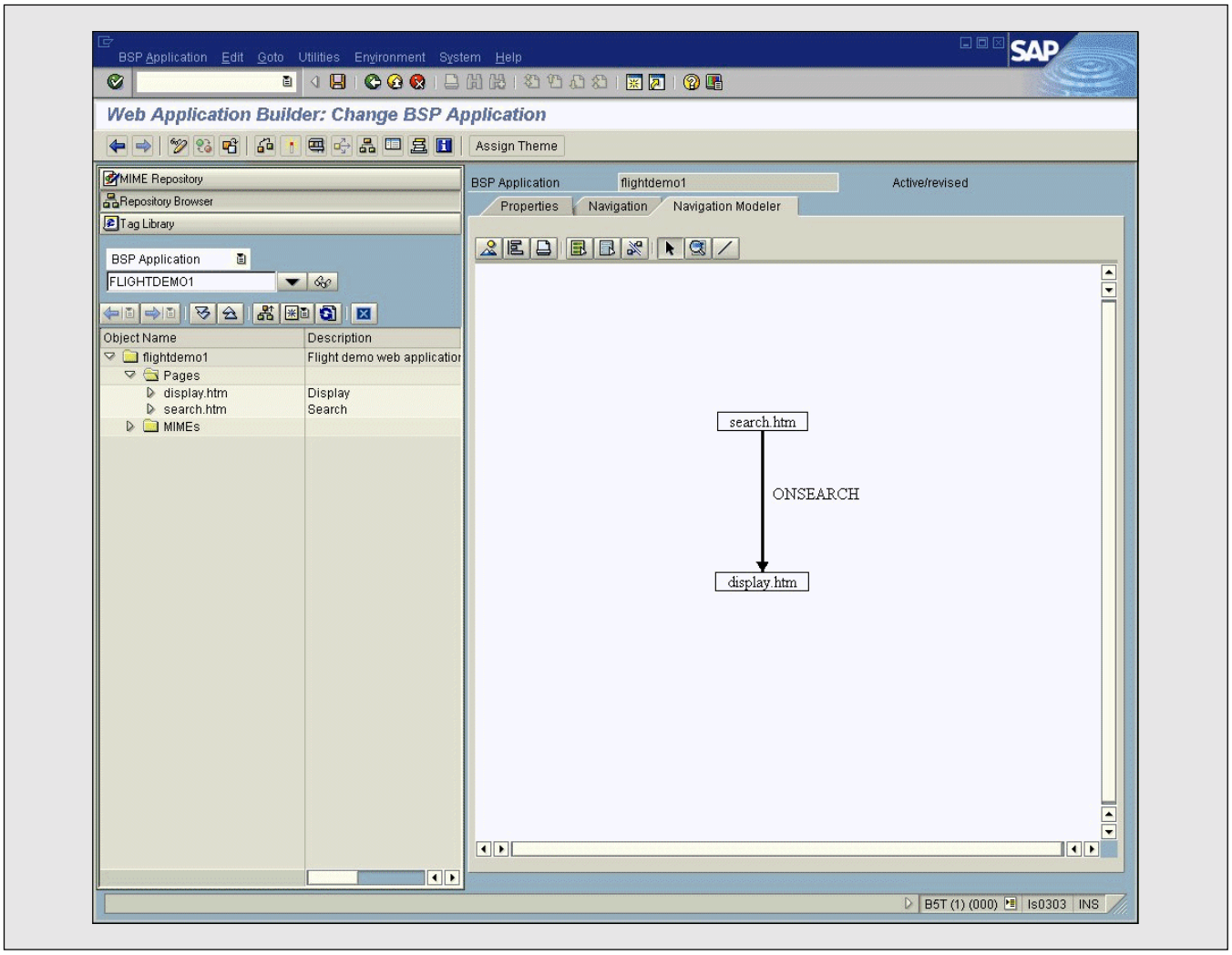
7. Return to the "Properties" tab of your BSP application ("Flightdemo1") and update the "Theme" field with your recently created theme name ("THM1"). This will be the default theme selected when the application is launched.

8. Re-launch your application, and the images or MIME objects you've added will be displayed.

Navigation Modeler

In creating our demo web application, we learned how to link Business Server Pages by using a Navigation object (refer back to Figure 14). This is a rather low-level approach, as direct references to web pages are buried into the code. If you later decide to dramatically change your application's flow, it will be difficult to locate these hard-coded references for updating. The Navigation Modeler can help.

Figure 23 The Navigation Modeler Graphical Display



The Navigation Modeler is a graphical tool that displays the relationship between BSP pages by means of "navigation requests." Pages are represented as nodes, and navigation requests as connectors from one page to another. Then, rather than referring directly to static page names in your code, you issue symbolic navigation requests. The system determines which physical target page is associated with a logical navigation request, and transfers control as desired.

Continuing with our demo application, let's employ this new technique by performing the following steps:

1. Double-click on your BSP application's name ("Flightdemo1").
2. On the "Navigation" tab, select the "search.htm" page for the field "First Page" by pressing "F4."
3. Select the "display.htm" page for the field "Target Page" by pressing "F4."
4. Specify a name for the "Navigation Request" field (e.g., "ONSEARCH").
5. Select the "Navigation Modeler" tab. Your application's logical model is displayed visually via an embedded graphic editor (see **Figure 23**).

> ✓ ***Tip***
>
> *You can manipulate navigation requests from within the graphic editor of the "Navigation Modeler" tab, and you can add new ones using the connection tool () on the toolbar.*

6. Finally, replace the hard-coded page navigation reference in your "search.htm" page. Open the page and modify your current `navigation->goto_page` statement so that it reads:

```
navigation->next_page( 'OnSearch' )
```

The application should function no differently than before the change.

Conclusion

The Web Application Builder is SAP's web application development tool of choice for developing Business Server Pages applications for the SAP Web Application Server platform. ABAP programmers will find developing with the Web Application Builder very comfortable since it uses the same editing, navigation, and debugging facilities as ordinary ABAP programming. Powerful tools like the BSP Editor, the Tag Library browser, and the MIME Repository help you quickly build applications, and also help enable the use of third-party design and development tools through WebDAV.

As a strategic development platform, the SAP Web Application Server combines modern web technology and the power and integration of mySAP.com component systems with unprecedented integration and flexibility. The Web AS complements SAP's current offering, the Internet Transaction Server, in enabling customers to develop powerful web-based applications to simplify and streamline interaction with both mySAP.com and non-SAP enterprise systems.

For more information on how to obtain and install the SAP Web Application Server, visit the SAP Service Marketplace at **http://service.sap.com/technology**.

Karl Kessler studied Computer Science at the Technical University of Munich, Germany. He joined SAP AG in 1992 as a member of the Basis modeling group, where he gained experience with SAP's Basis technology. In 1994, he joined the product management group of the ABAP Development Workbench. Since 1997, Karl has been Product Manager for SAP's business programming languages and frameworks. He can be reached at karl.kessler@sap.com.

