

Frédéric Heinemann, Christian Rau

Web Programming in ABAP® with the SAP® Web Application Server



SAP PRESS

Contents

Preface	11
1 Introduction	13
2 Overview: SAP Web Application Server	17
2.1 SAP NetWeaver	19
2.1.1 People Integration	22
2.1.2 Information Integration	23
2.1.3 Process Integration	24
2.1.4 Application Platform	25
2.1.5 Composite Application Framework	26
2.1.6 Lifecycle Management	27
2.1.7 Security	28
2.1.8 Interoperability	29
2.2 An Overview of the SAP Web Application Server	30
2.2.1 What Is the SAP Web Application Server?	30
2.2.2 The Architecture of Web Application Servers	31
2.2.3 The Architecture of the SAP Web Application Server	37
2.2.4 Characteristics of the SAP Web Application Servers	44
2.2.5 Applications of the SAP Web Application Server	47
2.3 The Internet Communication Manager	48
2.3.1 ICM Architecture	48
2.3.2 The HTTP Plug-in for the ICM	52
2.3.3 The SMTP Plug-in	54
2.3.4 The ICM Server Cache	56
2.4 The Internet Communication Framework	60
2.5 The J2EE Application Server	68
2.5.1 The J2EE Architecture	68
2.5.2 Characteristics of the SAP J2EE Engine	70
2.5.3 Combining ABAP and Java	72
2.5.4 Integrating ABAP and Java	74
2.6 Developing Web Applications	75
2.6.1 ABAP	75
2.6.2 Java	77

2.7	Web Services	79
2.7.1	Foundations	80
2.7.2	The Web Service Architecture	85
2.7.3	Developing Web Services	86
2.8	Security	89
2.8.1	Logon Procedure	91
2.8.2	Load Balancing	96
2.9	The Role of the Internet Transaction Server	98

3 Basic Principles: BSP Applications 105

3.1	Introduction and a Look at the Web Scenario Being Developed	105
3.2	Introduction to Languages and Standards	109
3.2.1	ABAP	110
3.2.2	DHTML	114
3.2.3	HTTP and HTTPS	154
3.2.4	XML	156
3.2.5	Cookies	160
3.3	BSP Applications	164
3.3.1	Components	165
3.3.2	Access to a BSP Application	175
3.3.3	Event Handler-Controlled Processing	182
3.3.4	Model View Controller Design Pattern	194
3.3.5	Web Dynpro	200
3.4	Including Mobile Clients	203

4 Development: Tools 209

4.1	Object Navigator	209
4.1.1	Introduction	209
4.1.2	Configuration	215
4.1.3	Repository Browser	218
4.1.4	Repository Info System	222
4.1.5	Transport Organizer	223
4.2	Web Application Builder	223
4.2.1	Editor	224
4.2.2	Version Management	229
4.2.3	MIME Repository	231
4.2.4	Tag Browser	234
4.2.5	Theme Editor	234
4.3	Service Maintenance	238
4.3.1	Services	238
4.3.2	HTTP Debugging	249
4.3.3	Trace	250
4.3.4	Runtime Analysis	252

4.4	WebDAV Interface	254
4.4.1	WebDAV as a Mediator Between the Worlds	255
4.4.2	Creating a Web Folder	259
4.4.3	Creation and Management of a BSP Application Layout Using Adobe GoLive 6.0	263
4.5	BAPI Browser	268
4.6	Online Text Repository	271
4.7	The Transformation Editor	275
4.7.1	XSLT	277
4.7.2	Simple Transformation	278

5 Practical Exercise: Creating BSP Applications 283

5.1	The First BSP Application	284
5.1.1	Creating a BSP Application	284
5.1.2	Creating a BSP Page	288
5.1.3	Graphic Objects	293
5.2	Server-Side Scripting	295
5.3	Page Fragments	298
5.4	Data Retrieval	302
5.4.1	Scenario 1: Customizing the Start Page	307
5.4.2	Scenario 2: Displaying Flights	312
5.5	Processing User Entries and Navigation	317
5.5.1	Event Control	318
5.5.2	URL Parameter	325
5.5.3	HTML Form Control	326
5.5.4	Enhancing the Flight Booking Portal	327
5.6	The Application Class	332
5.7	Formatting the Output	339
5.8	Multilingual Capabilities	344
5.9	Dictionary Services for BSP Applications	350
5.10	Checking and Handling Entries	358
5.10.1	The MESSAGE Object	358
5.10.2	Client-Side JavaScript	363
5.11	State Models	369
5.11.1	Hidden Form Fields	372
5.11.2	Client-Side Cookies	373
5.11.3	Server-Side Cookies	376
5.11.4	Enhancing the Flight Booking Portal	379
5.12	BSP Extensions	381
5.12.1	Using BSP Elements	382
5.12.2	Modifying BSP Elements	398
5.12.3	BSP Extensions Creation	402

5.12.4	Generating BSP Elements	403
5.12.5	Composite Elements	407
5.12.6	Scenario Executing the Booking	408
5.13	Public and Protected Access Areas	415
5.13.1	Applications with Public and Protected Areas	415
5.13.2	Enhancing the Flight Booking Portal	416
5.14	Model-View-Controller-Design Pattern	422
5.14.1	Creating the Controller	423
5.14.2	Processing Flow	426
5.14.3	Creating a View	428
5.14.4	Calling the View	429
5.14.5	Creating a Model Class	431
5.14.6	Calling the Model Class	434
5.14.7	Enhancing the Scenario	435
5.15	BSP Extension Expressions	439
5.15.1	BEEs for Flexible Code Listings	440
5.15.2	Using the Iterator tableView	457
5.16	Request Handler	466
5.17	SAP Web Application Server As Client	470
5.18	Web Services	473
5.18.1	Web Service Creation and Publishing	474
5.18.2	Calling a Web Service	494
5.19	Additional Functions	505

A Reference: Web Development on the SAP Web Application Server 509

A.1	BSP Extensions	509
A.2	The HTTP Interface of the ICF	516
A.2.1	IF_HTTP_EXTENSION Interface	516
A.2.2	IF_HTTP_SERVER Interface	518
A.2.3	IF_HTTP_CLIENT Interface	521
A.2.4	IF_HTTP_ENTITY Interface	524
A.2.5	IF_HTTP_REQUEST Interface	529
A.2.6	IF_HTTP_RESPONSE Interface	530
A.2.7	IF_HTTP_UTILITY Interface	531
A.2.8	IF_HTTP_STATUS Interface	532
A.2.9	IF_HTTP_HEADER_FIELDS Interface	534
A.2.10	IF_HTTP_HEADER_FIELDS_SAP Interface	536
A.2.11	IF_HTTP_FORM_FIELDS_SAP Interface	540
A.2.12	IF_HTTP_PROXY_CONFIG Interface	541
A.3	Interfaces and Classes for BSP Development	541
A.3.1	IF_BSP_APPLICATION Interface	542
A.3.2	IF_BSP_APPLICATION_EVENTS Interface	543
A.3.3	IF_BSP_NAVIGATION Interface	544
A.3.4	IF_BSP_RUNTIME Interface	547

A.3.5	IF_BSP_PAGE Interface	549
A.3.6	IF_BSP_PAGE_CONTEXT Interface	551
A.3.7	IF_BSP_SERVICES Interface	552
A.3.8	CL_BSP_MESSAGES Class	553
A.3.9	CL_BSP_GET_TEXT_BY_ALIAS Class	555
A.3.10	CL_BSP_CONTROLLER2 Class	556
A.3.11	CL_BSP_SERVER_SIDE_COOKIE Class	558
A.3.12	CL_BSP_MIMES Class	560
A.3.13	IF_CLIENT_INFO Interface	560
A.3.14	CL_HTMLB_MANAGER	569
A.3.15	CL_HTMLB_EVENT	571
A.4	Supported MIMEs	572
A.5	BSP Directives	574
A.6	Logging in the ICM	574

B	Glossary	579
----------	-----------------	------------

C	Sources and Further Reading	587
----------	------------------------------------	------------

	About the Authors	589
--	--------------------------	------------

	Index	591
--	--------------	------------

Preface

The SAP Web Application Server (SAP Web AS) forms the basis of all mySAP Business Suite components. It is the result of the continuous redevelopment of the original SAP Application Server.

Due to its vital importance, you must familiarize yourself with the functions, the architecture, the security aspects, and even the software development and maintenance of the SAP Web AS.

We wrote this book in order to support you in this task and in so doing, we can draw on our many years of experience as developers and consultants for an international business consultancy and on our work at SAP. We developed one of the first application systems based on the SAP Web AS to go into productive usage. Since then, we have assisted customers in implementing numerous Web projects. In this book, we would like to pass on the knowledge we gained from our experiences.

Amendments to the second edition

In the second edition, all notified mistakes in the first edition were corrected. We have also revised and amended the text that pertains to SAP Web AS 6.40. In addition to these general improvements, the following topics were added:

- ▶ Overview of SAP NetWeaver
- ▶ Web Services
- ▶ Business Extension Expressions (BEEs)
- ▶ Transformation Editor
- ▶ Introduction to Web Dynpro
- ▶ Log-on Scenarios

Credits

The journey from the conception of a book to its actual completion is a long and often complicated one. Many people helped us in this process. Please excuse us if in the following sections we do not mention a specific person even though he or she contributed to the success of this book.

We would like to thank countless numbers of our colleagues without whom this book would not have been produced. They were very understanding and helped us in significant and sometimes unknowingly, in subtle ways.

A book such as this always requires technical support. In this respect, we would especially like to thank Dirk Feeken of SAP AG for making resources available and Rüdiger Kretschmer, Steffen Knöller, and Jürgen Opgenorth for reviewing the manuscripts and answering our technical questions.

As much as this book is dependent on its authors and technical support, it is also indebted to its publisher; otherwise, all efforts would have been in vain. Therefore, we must thank Stefan Proksch from the SAP PRESS editorial office and the never-tiring engine behind our desire to be published, Florian Zimniak, whose patience, perseverance, and understanding helped in producing this book. For the English edition we would especially like to thank Nancy Etscovitz from UCG for copy-editing the book.

We would like to thank everyone, especially those close to us, who were not mentioned, but are worthy of recognition.

And last but not least, we would like to thank our countless readers who, because of their strong encouragement and enthusiastic response, made this second edition possible.

Many thanks!

Wiesbaden, May 2005

Frédéric Heinemann

Cologne, May 2005

Christian Rau

3.3 BSP Applications

BSP applications are standalone Web applications with presentation, workflow and application logics that are self-contained in terms of function. In many ways, BSPs are similar to the server page technologies (xSP) from other software manufacturers, such as Active Server Pages (ASP) from Microsoft and Java Server Pages (JSP) from Sun Microsystems. This technology has become relatively widespread in the field of Web development thanks to its various advantages.

Complete integration

BSP applications are developed on the SAP Web Application Server using the Web Application Builder. This is integrated into the SE80 development transaction. BSP applications are developed on the Sap Web aS using the Web Application Builder, which is integrated into the SE80 development transaction.

The presentation level of this type of BSP application where the actual display takes place (in this case on the client browser) is formed from a sequence of Web pages. This consists of the following elements:

► Static Web pages

These Web pages do not contain any sever-side scripting.

► Dynamically generated Web pages

These Web pages contain server-side scripting and are assembled to form a finished Web page only on request during the runtime of the application server.

► MIME objects

These objects include graphics, symbols, sound files, and style sheets, which can be included in Web pages.

In addition, client-side JavaScript for dynamic actions without server roundtrip⁹ and client-side cookies for the temporary storage of information can play a role when required.

So that the Web application can actually run on the client, a series of components and mechanisms is required. These interact with one another on the application server and need to be applied.

This chapter describes the individual components, the management and the processing sequence of these BSP applications. Also discussed is the

⁹ *Server roundtrip* means that data is exchanged with the application server to process interactions. Depending on the browser and coding, this may not be absolutely essential.

Model-View-Controller-Design-Pattern, which offers an interesting alternative to the BSP programming previously described.

3.3.1 Components

Underneath the presentation layer described above- that is, on the application server- are the individual parts of the BSP application. During run-time, these parts are processed and sent to the client in a format it can understand. Of course, the client's response can also be processed within the BSP application. A BSP application comprises several or all of the following components (see also Figure 3.9):

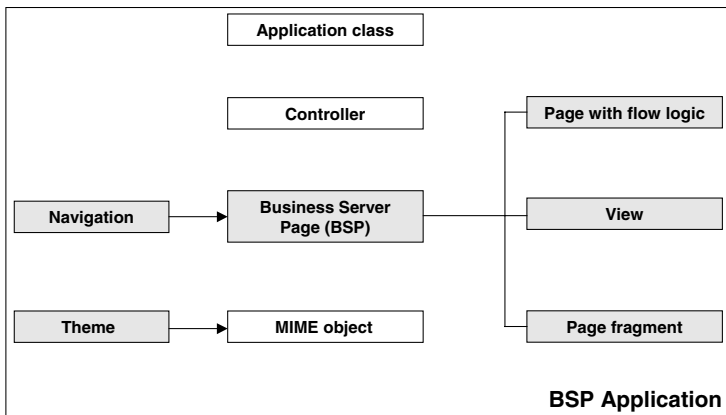


Figure 3.9 Components of a BSP Application

Naturally it is possible to use the development environment to link several static HTML pages using hyperlinks. These links can be invoked by the user in a more or less fixed sequence. However, this would be a very simple application that, with a little practice, could be treated using a basic text editor. It isn't necessary to use the SAP Web AS for this, and a small amount of Web space with an Internet provider would be perfectly adequate. However, you would quickly find this rather limiting. As soon as it becomes necessary to write back data and to use dynamic content (such as that generated from databases), the separation of layout and data, as well as other dynamic elements (such as for input checks and selection help) is unavoidable.

Static HTML

For this reason, you must initially divide the logical unit of a BSP application into presentation components, screen sequence control and application logics. A BSP application also contains a variety of management attributes.

In the actual conversion process, a clean division is not always possible, of course. When using BSP pages, for example, it is almost always necessary to integrate a flow logic into the layout. This one situation is avoided by using MVC Design Pattern. Because of the special nature of the MVC concept, a special section is devoted to this subject (see Section 3.3.4); so we will not cover the MVC components in any detail at this point.



As part of the BSP application, all the objects described below are integrated into the SAP correction and transport group (CTG) and are handled as a logical unit. This means that all objects in a BSP application can be fully and consistently transported between SAP systems.

Properties/Management Attributes

Every BSP application has a range of management attributes describing its general properties. This includes assigning the package, theme and application class, the HTTPS flag etc. These properties are defined on the Properties tab of the BSP application. Chapter 5 covers these points in detail.

Presentation Components

This section describes the components used to generate the graphical display on the screen.

BSP Pages

The BSP pages form the foundation for the contents ultimately displayed on the client browser. They may contain static Web code¹⁰ and dynamic scripting code (ABAP). This scripting code is transformed into code that the browser can understand (e.g., HTML) at the time that it is generated or when processing on the server begins. This makes it possible to specify the final appearance of the page at the runtime stage, i.e. at the time of the request.

A BSP page can contain the following:

► Pages with flow logic

The pages with flow logic are those pages whose process flow is controlled by event handlers. Normally there should be little application logic in the pages. Instead a special construct the application class, handles the application logic. In addition to the layout, a page with

¹⁰ Although only HTML code is used in the examples below, this does not mean that other standards such as XML, WML, and XHTML cannot be used.

flow logic implements the flow logic via the event handlers and has page attributes and type definitions.

Page attributes are global variables available on a BSP page. They are available in the layout and in all event handlers and "remember" their value across the entire runtime of the request. The content of a page attribute filled in the event can therefore be output in the layout code easily. Page attributes where the property `Auto` is activated are automatically filled with the value from the parameter of the same name after the page object is created, assuming a parameter has been transferred to the page. Page attributes can adopt any elementary type (except for `XSTRING`), structure, or table type.

Page attributes

Any type definitions can be generated in the BSP page type definition. The page can access these at any time. For example, the type of an internal table can be defined to specify the page attribute type.

Type definitions

The pages of type "page with flow logic" are executable and can be addressed via a URL or via navigation from another page. Chapter 5 covers the relevant programming work in detail. A complete BSP application can also be constructed solely of pages with flow logic and the relevant event handlers, as required.

► **Page fragments**

Page fragments are created in the same way as normal BSP pages but are referred to as page fragments. They differ from normal pages only in that they do not have separate event handling, no type definition and no separate attributes. They are integrated by other BSP pages using the `include` directive as a simple text-include. Page fragments inherit the page attributes from these other BSP pages.

Page fragments can in turn themselves include page fragments, but not pages with flow logic. Because page fragments enable the modular structure of the layout of BSP pages, they can make program code reusable.



► **View (MVC)**

Views are used to visualize data that is made available as part of the Model View Controller Design Pattern (see Section 3.3.4). Views are called almost exclusively by controllers, which means that they do not have a separate URL. Like pages with flow logic, Views have page attributes. Unlike these pages, however, views cannot be filled automatically when the `Auto` flag is called. The assigned controller is

responsible for filling attributes, for accepting requests as well as for filling attributes, accepting the request, and managing the flow control.

MIME Objects

MIME (*Multipurpose Internet Mail Extensions*) is an extension of the original Internet email protocol permitting the exchange of different data types on the Internet. These include audio, video and graphical data, style sheets, application programs and ASCII files. Client browsers are able to process these object types either using plug-ins or using integrated applications. This means that common browsers, for example, can display most graphics formats without external utilities. Other objects types, such as Flash animations, require plug-ins.



With each newly created BSP application, a directory of the same name is created in the MIME repository. This is used to store all the MIME objects specific to the application. This repository is used to manage the MIMEs centrally.

Themes

Themes are used to define replacements for MIME objects being used. This means that the appearance of BSP applications can be changed later on without having to change the layout source text. Each MIME object within a BSP application can be replaced by another object from the local file system. The theme concept is a simple way of customizing the layout of pages in a BSP application, even after it has been created.

A theme is created as a separate development object in the Web Application Builder (WAB) and acts as the container for all replacement definitions. For the changes to have an effect during runtime, the theme must be explicitly assigned to the relevant BSP application.

Replacement during runtime

Once a page requests a MIME object, the BSP runtime environment determines whether or not the running BSP application is assigned a theme. If this is the case, the correct object is determined from the replacement definition. This object is then transferred to the client in place of the object entered in the BSP application while the BSP application is running. In Figure 3.10, the MIME object to be sent to the client *LOGO.GIF* would be replaced during runtime by *MY_LOGO.GIF*, if the theme of the active BSP application was assigned. The *Themes editor* is available for working with themes.

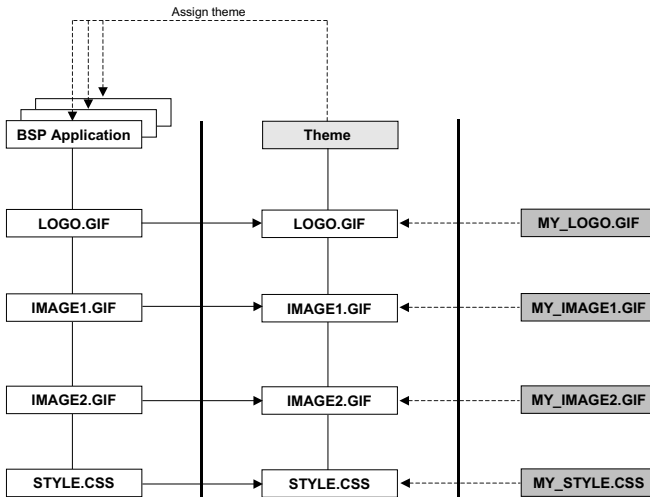


Figure 3.10 Replacing MIME Objects Using a Theme

BSP Extensions

When creating a BSP application, there is always the issue of how to ensure that the corporate design of this type of application is maintained. Cascading Style Sheets are a well established concept. However, with larger projects involving several developers there is the risk that the defined style instructions will not be used correctly. In the worst case scenario, the appropriate CSS elements may need to be continually re-assigned for each HTML element on each BSP page. This is a lengthy and error-prone process. Not least, the clarity of the HTML code also suffers, making changes even harder to make later on.

BSP extensions can help here. They represent an abstraction technique that can be used to simplify both the syntax and the semantics of HTML code blocks. This mechanism has an open structure and can be used in sections of XML and WML code, for example.

A BSP Extension is a container for BSP elements. Each element is assigned an ABAP class in the BSP context. Embedded in this class is the functionality to create the code to be executed on the client-side and its functionalities. The definitions of the elements and the mapping to the ABAP classes are flexible. This technology can be used to meet a variety of requirements not limited to graphical objects.

Container for
BSP elements

The SAP Web Application Server provides an infrastructure for managing BSP extensions. For this reason, these can be used relatively easily as part of BSP applications. Release 6.20 of SAP Web AS includes many prede-

Extension
infrastructure

financed extensions, such as HTML Business for BSP (HTMLB). These extensions can be expanded in any way and can be modified for specific purposes. Naturally you can also develop your own BSP extensions. The BSP extensions can be created and edited using the editor integrated into the development environment.



By using HTMLB as a template for creating your own extensions you can save yourself both time and effort.

BSP extensions can be used in BSP pages via the `extension` directive. In addition, composite elements can be created to unite multiple BSP elements into a sub-quantity. This makes it possible to affect all the BSP elements simultaneously when changing the layout, which reduces the amount of work involved in complicated BSP applications. This reduces the amount of work involved in complicated BSP applications.

Each BSP extension consists of a collection of BSP elements. Each of these elements has defined attributes and is assigned to an ABAP class. The attributes provided in the element represent the input parameters for the assigned ABAP class and are used to affect the appearance, the input behavior and other functionalities. BSP elements are inserted into BSP pages in XML notation.

In the outgoing HTML data flow, the element class writes the serialized HTML coding on the basis of the functionality provided by the element. It is assumed here that all elements in an extension support a common output style.

Advantages of BSP extensions

Using BSP extensions has the following advantages:

- ▶ The HTML code only needs to be developed once. Changes have an immediate effect on all calls of the elements. This also applies across applications. This increases the reusability in terms of corporate design.
- ▶ The ABAP class (element class) assigned to the element may contain additional logic to generate browser-dependent HTML code. This avoids browser-dependent coding in the layout.
- ▶ The style sheet assignments are also located in the element class. The fact that CSS assignments take place at *one* specific point ensures that the generated HTML coding places the correct references on the style sheets.
- ▶ The standard XML syntax can—unlike HTML coding in the layout—be parsed and checked at the point of generation. This avoids errors.

As well as a BSP extension for standard HTML elements such as buttons, input fields, dropdown lists, etc., highly specialized extensions can also be implemented. An example of such an extension could contain a composite element to implement a complete newsticker including generation frame and application logic. The newsticker can then be made available in any BSP applications via the directive extension with all its attributes parameterized.

Components of the Flow Control

A program's flow control determines the temporal and logical flow of an application. The time when specific components of a program are executed is partly fixed in the case of BSP applications and can partly be controlled by the developer. The first group includes the event handlers that are part of BSP pages with flow logic. The second group include the navigation structure and controller, for example.

Event handlers

The event handlers are executed at specific points during the runtime of a BSP page in a fixed, defined sequence. They are filled with ABAP code and permit access during runtime to specific objects such as the application class or to specific runtime objects in order to provide access to request information. This topic is covered in detail in Section 3.3.3.

Navigation structure

The navigation structure is used to define navigation requests. These describe the start and destination of a request, i.e. the page order in the navigation process. Assigning the pages via navigation requests creates a purely formal description of the navigation scheme within a BSP application. This makes it possible, for example, to change the flow control of a BSP application without intervention in the coding.

Controllers (MVC)

Controllers are another part of the MVC Design Pattern. From the data of an incoming request, controllers evaluate a suitable view based on a model. This can then be rendered for the response. They represent the link between the Model and the View.

Components of the Application Logic

The application logic (business logic) handles the actual processing of the data. As is the norm in standard ABAP programming, the application logic can be addressed in the form of BAPI's, function modules, or class libraries from a BSP application, for instance from the event handlers.

The SAP Web Application Server provides additional structuring aids—the BSP application class and the MVC—that can be used to encapsulate the required application logic.



It is even possible to house the application logic in the layout part of a BSP page. We can only advise against this, however!

Encapsulation of the application logic

The application class is used to encapsulate the application logic of a BSP application. This is realized using a standard, global ABAP class. It calls business data from backend systems using BAPI for example, and writes this data back to the BSP application after processing. An application class of this type is assigned to the BSP application and is then directly available to each page of the BSP application with its components (attributes, methods, etc.) via the standardized object reference `application`. This happens automatically. Neither "manual" declaration nor instancing is required before use.

The tasks of a BSP application that are implemented in application classes can include the following.

- ▶ Cross-page saving of BSP application data in the form of attributes
- ▶ Encapsulation of the application logic in methods
- ▶ Framing of repetitive tasks (e.g. checking of authorizations, complex input checks, saving and restoring data using server-side cookies) in methods

An application class can be used in any number of applications as well as in typical ABAP programs. However, only one application class may exist per BSP application. The assignment is made on the Properties tab of the BSP application, as shown in Figure 3.11.



Other classes can be included in a BSP application in addition to the application class.

Existing application functionalities of the SAP Application Server or the backend systems are normally framed within the application class. This makes the coding clearer and easier to maintain.

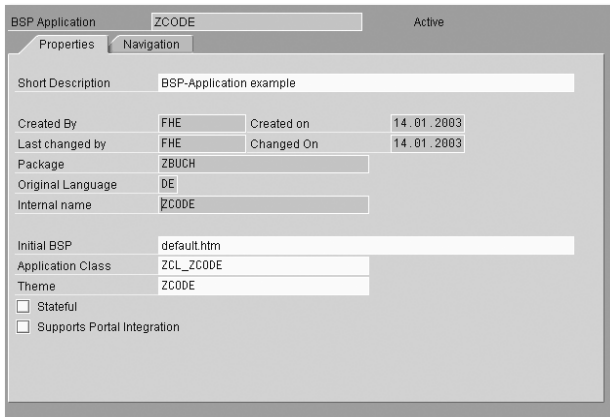


Figure 3.11 Assigning the Application Class

When using application classes, the following should be noted:

- ▶ If the application class is assigned a stateful BSP application, its life cycle is exactly the length of the BSP application. This makes it ideal for data storage.
- ▶ If the application class is assigned a *stateless* BSP application, its life cycle is exactly the length of a BSP page, i.e. from receipt of the request to the completion of the `response` object. It is not suitable for data storage as it is destroyed directly afterwards, like all objects. In this case, server-side cookies are normally used for data storage.
- ▶ Application classes are singleton. There can only be one instance per session.
- ▶ A BSP application can be assigned only one application class.
- ▶ The constructor must be free of parameters.

You should carefully consider how you will implement data acquisition. For example, it is inefficient to acquire the complete material master of the backend system in the application class in a stateless BSP application. This process would then take place at each page change and the performance would be permanently slowed. In a stateful application, however, it could be very useful as the data will have been loaded from the backend just once when the application is first called when the `application` object is instantiated.

The interface `IF_BSP_APPLICATION_EVENTS` can be implemented in the application class. This interface makes a BSP application's other processing times accessible via methods that permit both flexible control and

Application events

access to the processing logic. The interface's individual methods are presented in Appendix A.2.

Application base class It can be useful to derive the application class from the pre-defined base class `CL_BSP_APPLICATION`. This class provides methods that can be called or set: the session timeout, the current URL for the BSP application, the status mode, and so forth. The methods of the individual interface concerned will also be presented in Appendix A.2.

Example Figure 3.12 uses a simplified scenario to explain the process flow of a request with encapsulated application logic. The search for an address is triggered on HTML page 1. The request is directed to the BSP page HTML page 2 Using the `onInputProcessing` event. In this process the name is transferred as an Auto page attribute. The `onInitialization` event calls the application class that, in turn, controls an RFC call for data acquisition in the backend. The result is returned to the application class where it is made available from the event handler. The attributes now filled are output in layout code and appear in this form in the client browser as HTML page 2.

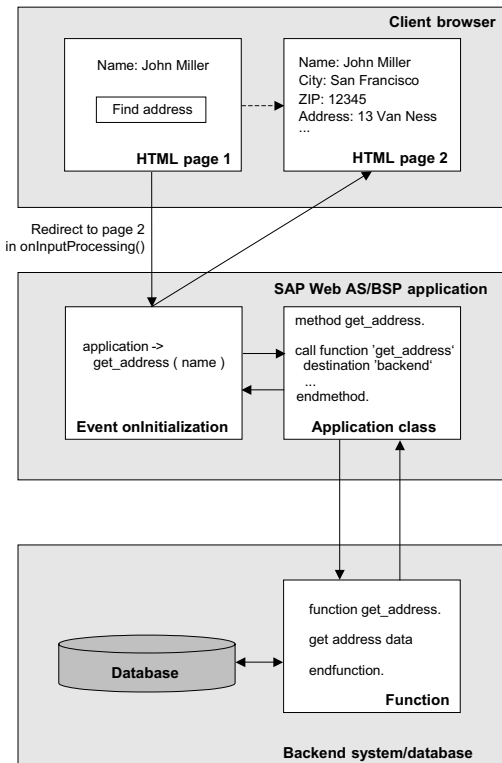


Figure 3.12 Typical Process Flow for Encapsulated Application Logic

Model (MVC)

The Model is used as an application object to control the behavior and the application data. It answers both information requests via its status, which generally comes from the View, and instructions for status changes which are usually sent by the Controller.

The Model recognizes neither its Views nor its Controller. This means that the model alone is used for internal data processing, without referring to the application with its user interface. In short a model is specified using a controller class reference.

3.3.2 Access to a BSP Application

This section explains the special access features of a BSP application. An overview of the standard error message is given in the section following it. The system-specific URL parameters are then presented. You can use these parameters to configure the application call and can modify the parameters to your own requirements.

Addressing

A BSP application is executed by an URL. You can call the application directly by inputting the URL in the address line of the client browser.

You can also save the application in your *Favorites* and call it easily using the favorites management feature. It can also be saved to the desktop just like normal links. In this process, name/value pairs (see below) can also be specified for parameterization. This means that the web application can be called after being parameterized.



The URL of a BSP application has the following general structure:

```
<Protocol>://<Host>:<Port>/<Name range>/  
<Application name>/<Page>?<URL Parameters>
```

Parts of the URL

► Protocol

BSP applications support the HTTP and HTTPS protocols. If HTTPS is to be used, this protocol must be available as an ICM service. Therefore it must be configured via instance profile and the corresponding system files must be installed.¹¹

11 We do not explain this procedure in this book. You can find the relevant installation instructions in the documentation for the SAP Web Application Server under "SAP Web Application Server—Using the Secure Sockets Layer Protocol" or on the Internet at <http://service.sap.com/instguides>. The installation package is available to authorized customers in the SAP Service Marketplace at <http://service.sap.com/swcenter> for using the SAP Cryptographic Library.

► **Host**

The "<host>" is the name of the application server on which the application is to be executed. Either the IP address of the host or the DNS name including network domain should be specified.

► **Port**

The port number comes next. It specifies the port on which the application is to be run, if the default value is to be circumvented. The relevant protocols are assigned to the ports via the profile settings (profile parameter `icm/server_port_<xx>`).

► **Name range**

The name range is the BSP application's name range code. SAP applications are supplied in the **sap** name range. BSP applications can be created in a separate name range.

► **Application name**

The application name is the name of the BSP application as defined in the development environment.

► **Page**

The page is the name of the required destination page of the BSP application. This may be a BSP page, a static page or a controller of the MVC. It's helpful to use the initial page to ensure that the application is also correctly initialized.

► **URL parameter**

The application can also specify parameters as name/value pairs when called. These may be specific to the system or may relate to the application (see at a later stage in this section). They are separated from the actual URL by a question mark (?).



The last two parts can also be omitted. In this case, the entry point is the default page set in the BSP application properties. Spaces are not permitted, but can be reproduced in the final URL with escaped URLs. Spaces are displayed as `%20` in this case.

A URL in the SAP name range may then look as follows:

```
http://www.my-webas.com:8080/sap/bc/bsp/sap/zcode/start.html?var_1=init
```



As these long URLs are rather unwieldy, you can define an external alias in the ICF. The alias then maps the address and the previous URL might then look as follows:

```
http://www.my-webas.com/zcode
```

Error Messages

If the BSP application cannot be reached or executed, the system sends a standard error response as an HTTP error code. The best known error is probably the HTTP error code 404, which is sent if the requested resource can not be found. The code 500 internal server error is especially critical during development. In appendix A.1 you will find a list of the error codes generated by the SAP Web Application Server.

To obtain meaningful error messages from the server, especially during development relating to error code 500, the **Show friendly HTTP error messages** option should be deactivated in Internet Explorer (see Figure 3.13). Otherwise, Internet Explorer displays "user-friendly" error messages, which have virtually no practical use during the development process.

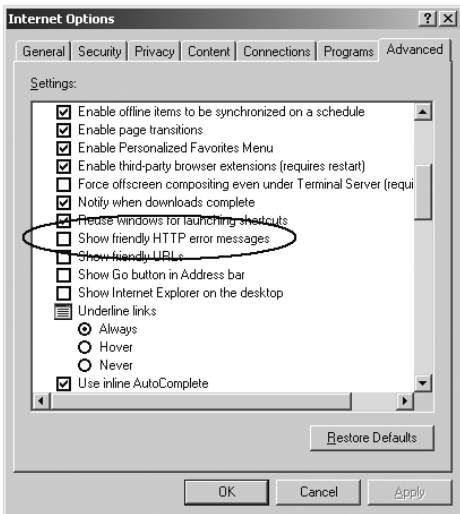


Figure 3.13 Displaying Meaningful Server Error Messages in Internet Explorer

The SAP Web AS also allows you to respond to logon and application errors and to call logon pages. There are two alternatives here: First you can create your own error pages which provide the user with comprehensive information on the situation occurring or suggested solutions. Or, you can use a redirect to forward the user to another URL. With a redirect, the form fields already transferred to the destination page can also be transferred to the new address (see Figure 3.14).

Response to logon and application errors



This permits an immediate display of the parameters such as those that cause the application to produce errors. This will save you time during long debugging sessions.

System-Supported Logon

BSP application
SYSTEM

The BSP application `SYSTEM` has been available in the ICF since the release of version 6.20. This application provides the user with a parameter-controlled, convenient logon template instead of the typical browser pop-up. In addition, the application features a logoff scenario for stateful applications, which immediately releases system resources that are no longer necessary.¹²

The introduction of SAP Web AS 6.40 has made the logon and logoff processes more convenient. You can see the new **System Logon** option with **Settings** button on the **Logon Error** sub tab in Figure 3.14.

Figure 3.15 displays the many setting options available. Logging onto the system by using the options described, creates the logon template displayed in Figure 3.16. This template is rendered directly in the browser window.

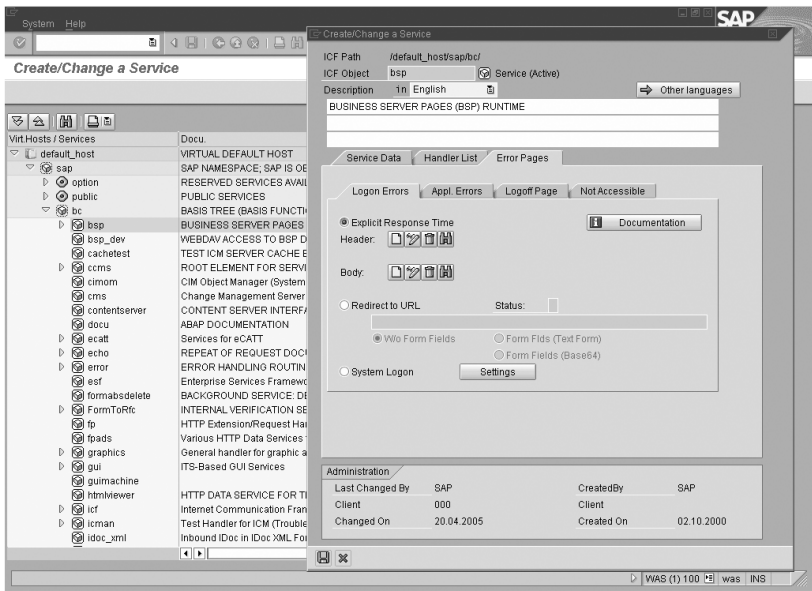


Figure 3.14 Creating Your Own Error Pages or Using a Redirect in the SICF Transaction

¹² For more information on the BSP application `SYSTEM`, please refer to OSS Note 517860.

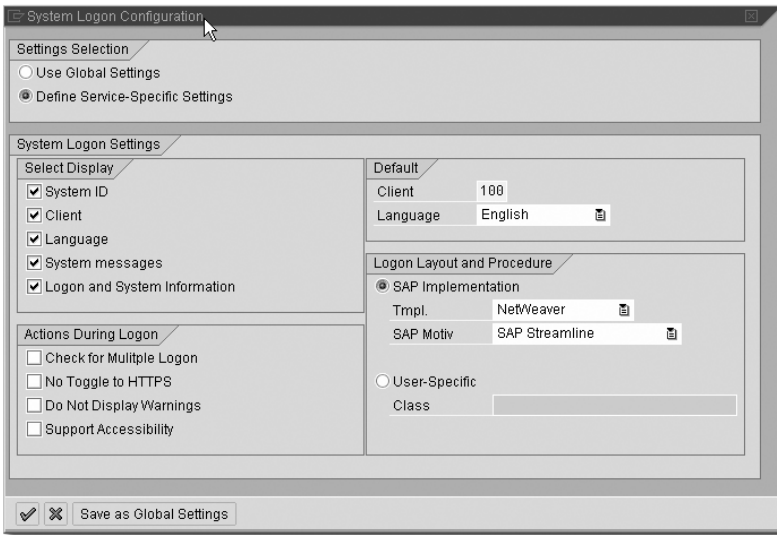


Figure 3.15 System Logon Setting Options

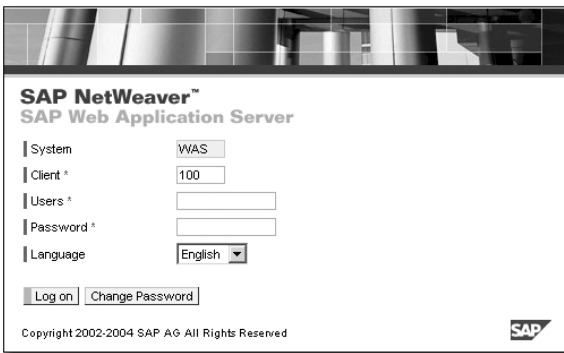


Figure 3.16 System Logon Screen

URL Parameter

A URL controls the behavior of a BSP application. To do this, this URL is expanded by a query string. The part of a URL designating a query string is preceded by a question mark (?).

The parameter name and its associated values are case-insensitive. One exception is the `sap-exiturl` parameter, if the reference is to a case-sensitive server.



The query string has a sequence of name/value pairs, which are separated by the commercial And (&), e.g.:

```
http://www.my-webas.com:8080/sap/bc/bsp/sap/zcode/start.htm?
status=0&org=sap
```

In this case the BSP application address has two added query string parameters added: `status` and `org` with the values "0" and "sap". This has an equivalent effect to the value assignment in the code using:

```
status = '0'.
org     = 'sap'.
```

System-specific URL parameters

The SAP Web Application Server is familiar with a range of system-specific URL parameters. They are automatically checked by the server on each call and some have a decisive effect on the application behavior.

The possible system parameters have the following basic structure:

```
sap-⟨parameter-name⟩=⟨value⟩
```



Multiple system parameters can be combined.

Given below is a description of the individual parameters. Their possible uses are also explained using examples.

► **sap-sessioncmd**

The "open" value is used to restart a running BSP application or, if it is not yet running, to start it for the first time.

```
http://www.my-webas.com:8080/sap/bc/bsp/sap/zcode/start.htm?sap-
sessioncmd=open
```

The "close" value ends a running BSP application. On closing, the browser is directed to an empty page. Closing the browser has the same effect as entering the transaction code `/n` in the SAP frontend.

```
http://www.my-webas.com:8080/sap/bc/bsp/sap/zcode/start.htm?sap-
sessioncmd=close
```

► **sap-exiturl**

The `sap-exiturl` parameter retrieves the URL address specified.

Combining this parameter with `sap-sessioncmd` makes the application easier to close. If the current BSP application is explicitly ended in the Web browser and you want to go to a destination page, the call could look something like this:

```
http://www.my-webas.com:8080/sap/bc/bsp/sap/zcode/start.htm?sap-
sessioncmd=close&sap-exiturl=logout_success.htm
```

► **sap-theme**

This parameter specifies the theme used for the BSP page called and

therefore affects the overall appearance of the application. A theme can be described as a collection of replacement definitions for MIME objects.

A theme defined as the default for a BSP application is therefore overridden.

http://www.my-webas.com:8080/sap/bc/bsp/sap/zcode/start.htm?sap-sessioncmd=open&sap-theme=vip_customer



► **sap-themeRoot**

This parameter defines paths to other locations where style sheets can be found.

► **sap-syscmd**

The `nocookie` value ensures that the session cookie is not saved on the client but is transferred as part of the URL instead. The session cookie is then concealed in the URL mangling code.

http://www.my-webas.com:8080/sap/bc/bsp/sap/zcode/start.htm?sap-syscmd=nocookie

► **sap-htmlb-design**

This value allows you to switch between different HTMLB design variants. `CLASSIC`, `DESIGN2002` and `DESIGN2003` are possible values.

► **sap-domainRelax**

This value initiates the use of domain relaxing.

► **sap-accessibility**

When this parameter is activated, an automatically activated checkbox appears on the logon screen to assist the visually impaired in entering details.

The system parameters below are used to control the various logon parameters on the SAP Web Application Server.

Logon parameters

► **sap-client**

This parameter defines the clients on which a logon to the SAP Web application server takes place. A client which is specified here overrides the pre-defined default client. If the client does not exist in the system, an error message will be issued.

► **sap-user**

The parameter `sap-user` ensures that the logon to the system is carried out under the name specified. If the user does not exist in the system, an error message will be issued.

► **sap-password**

The parameter `sap-password` transfers the password for logging the user on. If the password is not correct, an error message is issued.



The password should never be used in a URL query string. Firstly there is a transfer in plain text (HTTP) and secondly, these URL strings are stored temporarily in the cache and could in theory be accessed by third parties.

► **sap-language**

This parameter specifies the logon language. This means that e.g. another available language version of the BSP application can be loaded onto the system. The default logon language is then overridden.

An example of the entire logon process for an English-speaking user might look something like this:

```
http://www.my-webas.com:8080/sap/bc/bsp/sap/zcode/start.htm?sap-sessioncmd=open&sap-theme=vip_customer&sap-client=100&sap-language=en&sap-user=en_george
```



As the password is not part of the query string, this component is missing from the URL for complete authentication. A corresponding pop-up window will appear to add the missing data.

URL-mangling code

Another part of the URL is what is known as *URL-mangling code*. This code is generated by the server and is given in brackets. The coded values contain various logon, session, and theme settings for the called BSP application.

An example of mangling code might be:

```
http://www.my-webas.com:8080/sap(bD1kZQ==)/bc/bsp/sap/zcode/start.htm
```

3.3.3 Event Handler-Controlled Processing

The BSP application processing flow conforms to a permanently defined scheme. The general flow logic is therefore constant, regardless of whether it is for simple address management or a complex task management process with integrated availability check via a backend system. A BSP application normally consists of multiple BSP pages (pages with flow logic). The user starts the application on an entry page and then navigates through the application to the various BSP pages. At some point, the user

exits the application. The individual steps in this processing flow are (as in Figure 3.17):

- ▶ Start of the BSP application
- ▶ Creation and display of the requested BSP page
- ▶ Response to user inputs and navigation as required
- ▶ Exiting the BSP application¹³

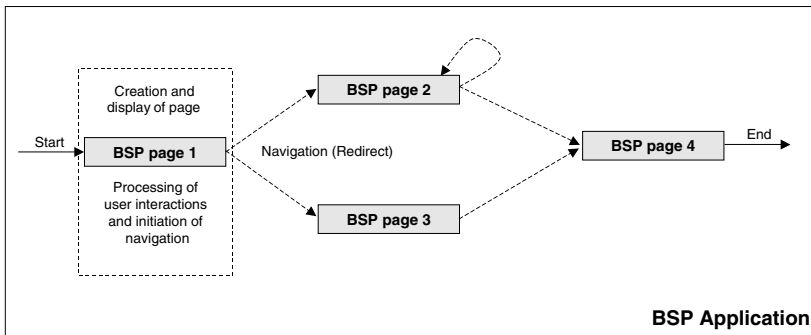


Figure 3.17 Processing Flow and Navigation in BSP Applications

These different steps are covered below in depth. Firstly two important aspects must be considered in more detail—the *event handler* and the *status model* used in the BSP application and the BSP page. These two points have a significant impact on the application processing flow. The event handlers have already been covered in brief in section 3.3.1. These handlers are available at specific times to process a BSP page. During these processing times, separate program logic can be executed to implement very specific tasks within the processing logic of a page. The processing flow, in particular running through the various event handlers, is also affected by the status model currently active for the application in question. The status models supported in the SAP Web Application Server are explained below.

Status Model for BSP Applications

There are two status models available for use in BSP applications: stateful and stateless. This functionality is provided by the ICF which supports both operating modes in the server role. A running BSP application is known as a *BSP session*. In section 3.3.2, we said that the start and end of

Stateful and stateless applications

¹³ Exiting a BSP application is only of interest in the stateful model. This stateful model is explained in more detail below.

a session can be influenced from outside by the user. The end of a BSP session can also be initiated by the application itself and by the Web browser being closed. The important thing to note here is the separation between the BSP and browser session. A browser session can only be ended by closing the browser.

Stateful A stateful BSP application is—as with a traditional SAP transaction with SAP GUI screens—executed across all user interactions in a single context (roll area). The application context is therefore maintained throughout the response. As the application continues to run, the corresponding context is rolled into the work process. This means that data entered by the user while running the application or that has been determined by the application itself, is maintained throughout the entire duration of the session. With stateful applications the data is stored by the application class. Figure 3.18 illustrates the stateful model. In this process a separate session is provided in the SAP Web Application Server for each session of a Web browser. This session contains the application context and is available for multiple request/response cycles. The dark blocks of different lengths in the graphics represent user activities. These are the points at which the resources of the servers are actually being used.

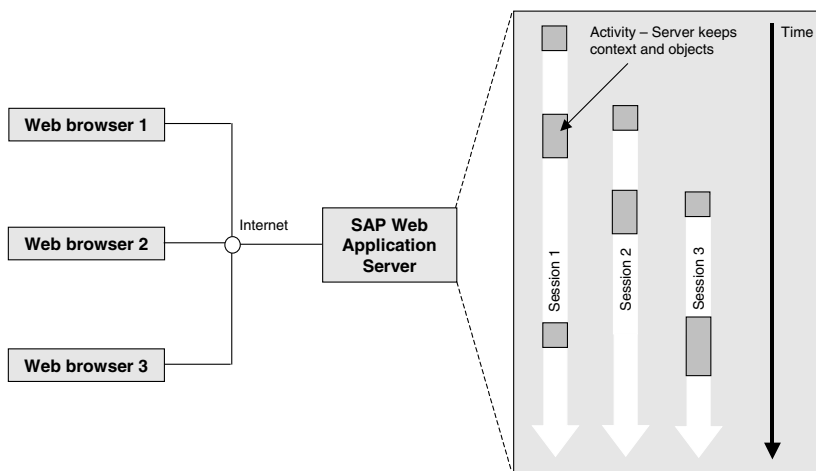


Figure 3.18 Stateful Model in the SAP Web Application Server

Session cookies The statelessness of the HTTP protocol is a problem when implementing this action. There is no implicit mechanism for assigning independent requests from a common logical session to a context, for example. The BSP runtime environment solves this problem using session cookies. To do this, a multi-digit session ID is generated and a unique stamp is added

to each request. This means that the requests can then be identified as part of a specific session. The name of the session cookie stored on the client side is `sap-contextid` and is valid to the end of a session (the browser session ID corresponds to the BSP session ID here). The assignment is made via the URL of the BSP application. This means that a BSP application can only be executed once at a specific time within a browser. Another BSP application has a separate session cookie and can be operated in parallel in the same browser. The same BSP application can be executed by multiple users and browsers any number of times.

The stateful model has a range of advantages and disadvantages to be considered:

Programming stateful BSP applications is relatively straightforward. Once data has been acquired at great cost, it can be stored in the attributes of the application class. On the next page, this data can simply be re-accessed. Costly database accesses can therefore be kept to a minimum. This means that it is not necessary to re-read large quantities of data, which could result in significant improvements in performance on the server side. In addition, the network load is minimized thanks to fewer requests to the backend systems.

Fewer database accesses

However, consideration needs to be given to the fact that it is very easy to distinguish between the state within the application and the state a user adopts on the basis of his user interface. A simplified example of this is using the browser button. These client-side actions are not necessarily forwarded to the server. Possible inconsistencies should be picked up on here and the client and server should be "re-synchronized."

However, the simplified programming means that each session of the relevant context needs to be saved. As the number of sessions is at least the same as the number of users (in the same way as the SAP GUI), a much greater load is normally generated on the SAP Web Application Server than with the stateless model. This places increased demands on the memory resource in order to be able to execute a variety of sessions in parallel. If the available memory space is exhausted no further users will be accepted, they will be rejected from the system! If the user does not explicitly end the BSP application, resources in the system are also blocked for unnecessarily long periods of time. If the user simply goes to another page, for example, the session in the SAP Web Application Server is maintained. In these cases, the Web browser does not automatically log off from the system. The context is retained here and is only released after a specific amount of time.

Memory resources



The time period for an auto-logout from the SAP Web Application Server is specified in a timeout parameter in the instance profile.

Stateless

Contrary to the stateful model, in the stateless model, SAP Web Application Server resources are not blocked for unnecessarily long periods of time. After processing a request, all occupied resources (application context) are released again immediately. A new application context (roll area) is created for each request and is rejected after the response.¹⁴ Resources are therefore directly used only during the processing of a request. This conservative use of the resources makes the stateless model ideal for implementing Web applications with a number of parallel accesses, because it provides good scaling of the SAP Web AS. This concept is illustrated in Figure 3.19. This shows that for each access to a Web browser, the resources for processing the request are only used for a short time on the server.

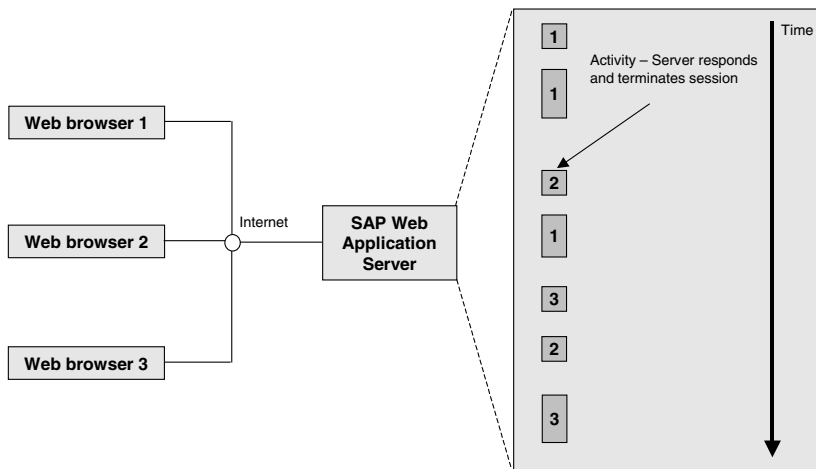


Figure 3.19 The Stateless Model in the SAP Web Application Server

The disadvantage of the stateless model is that in releasing the application context, data is read several times and needs to be prepared. The memory advantage is therefore unfortunately offset by runtime losses. The use of different techniques means that data can still be recovered in the stateless situation via requests. These include:

¹⁴ In stateless mode, the very first session ID is used to identify the associated browser session.

► **Hidden form fields**

These hidden form fields are transferred when sending a form, but are invisible to the user. These are input fields with the `type=hidden` attribute.

► **Client-side Cookies**

The data is temporarily stored in small text files; certain restrictions need to be considered (see Section 3.2.5).

► **Server-side cookies**

The data is not located on the client-side but on the server. Unlike client-side cookies there are no size restrictions (see Section 3.2.5).

► **DB tables**

Another option is to store the data in a DB table specifically created for this purpose. This allows you to standardize the tables as you wish, and increases the performance for accesses. The disadvantage is the increased amount of programming work.

Now that the two state models have been described, we'll explain how to set up your mode of preference. This can be set up either during development or during runtime. If nothing is set, the BSP application will always work in the stateless mode (default setting).

During development the required mode can be set up on the **Properties** tab of a BSP application (see Figure 3.20).

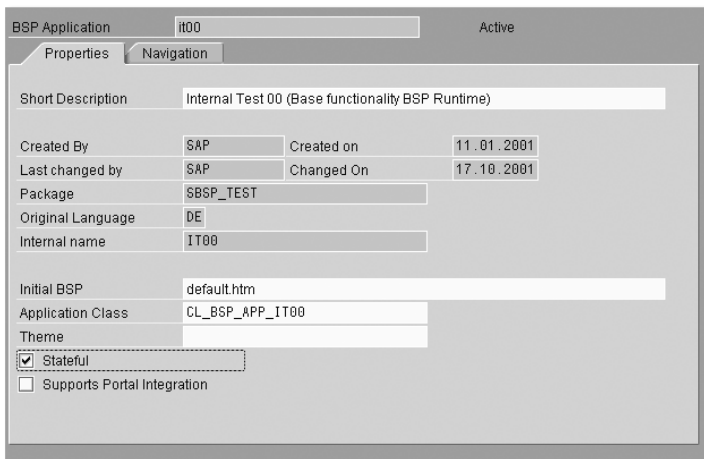


Figure 3.20 Setting a BSP Application to "Stateful"

However, individual BSP pages of a BSP application can be set to "stateless" or "stateful". This is illustrated in Figure 3.21. As you can see from the illustration, various settings can be made relating to the life cycle (in stateful mode). These include:

► **Until Page Change**

The page is destroyed after each individual request.

► **Request**

The page is destroyed after each individual request. In other words, it exists only for the duration of the session.

► **Session**

The page is destroyed at the end of the session.

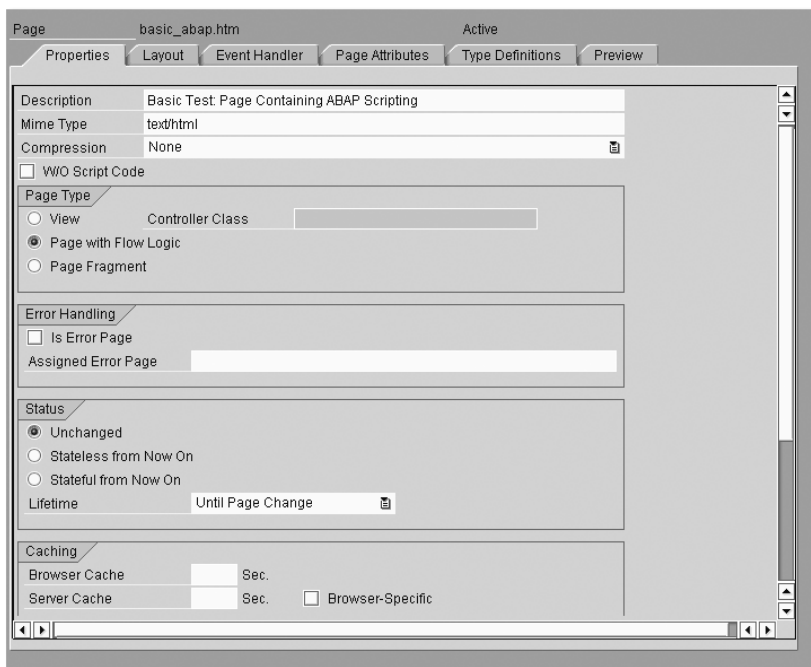


Figure 3.21 Setting BSP Pages to "Stateful" or "Stateless"

You can also switch between stateful and stateless during the runtime (dynamic). To do this there is the `runtime` object, relating to the `IF_BSP_RUNTIME` interface. The setting is made by setting the attribute value to `keep_context`. This attribute can adopt the values "0" and "1." The example below sets the stateful and stateless modes:

```
runtime->keep_context = 1. " set application to
                           stateful
runtime->keep_context = 0. " set application to
                           stateless
```

These settings override any definitions made from the development environment.

The Event Handlers

The event handlers represent the event concept of BSP pages. A range of pre-defined handlers passed through in a specified sequence when processing a page. These are:

► **onCreate**

This handler is called when a page is first opened, and is used for the one-off initialization of objects and data required on the page. The handler is always called when a BSP class is generated. The handler is called precisely once in stateful mode when generating the page. In the stateless mode, the page object is re-initialized each time—in other words, the handler is called anew each time. If you are using the stateful mode without explicit navigation and are simply running through the page once more, the page instance is retained. The handler is then not passed through again.

► **onRequest**

This handler is called at each access (request) to page. It is used to restore internal data structures from a request.

► **onInitialization**

This handler is used for data retrieval. Ideally this data is saved in page attributes and is then also available in the layout. In addition, any programs can be called. This handler is triggered after the `onRequest` handler.

► **onLayout**

This is a hidden event handler. No code can be added. The layout of the page is rendered and the HTTP data flow generated for the response.

► **onManipulation**

This event handler permits the subsequent manipulation of the http data flow. It is processed once the layout elements of the page have been generated. This handler is rarely used.

► **onInputProcessing**

The task of this handler is to manage user inputs, input checks, and navigation (on the same page or subsequent pages). Certain requirements need to be met for this handler to be triggered.

► **onDestroy**

This handler is called immediately before deleting a page instance. Follow-on actions can then be carried out here for a page. It is the opposite to the `onCreate` handler. In the stateless mode `onCreate` and `onDestroy` are passed through for each request/response cycle. In stateful mode this handler is not called for each cycle, only when the system changes to stateless mode.

There are certain global runtime objects available within each event handler. These objects permit e.g. access to the `request` object, the `response` object or permit navigation between the BSP pages via the `navigation` object. An overview of these available objects and their meanings can be found in Table 3.5.

Object	Meaning	Interface/Class/Type
<code>runtime</code>	BSP runtime information	<code>IF_BSP_RUNTIME</code>
<code>application</code>	Attributes and methods of the application class	separate class of derivative of <code>CL_BSP_APPLICATION</code>
<code>page_context</code>	Page context object ¹	<code>IF_BSP_PAGE_CONTEXT</code>
<code>page</code>	BSP page information	<code>IF_BSP_PAGE</code>
<code>request</code>	Access to the <code>request</code> object	<code>IF_HTTP_REQUEST</code>
<code>response</code>	Access to the <code>response</code> object	<code>IF_HTTP_RESPONSE</code>
<code>navigation</code>	Data transfer and navigation	<code>IF_BSP_NAVIGATION</code>
<code>event_id</code>	User interaction	<code>STRING</code>
<code>messages</code>	Treatment of error messages ²	<code>CL_BSP_MESSAGES</code>

1 The `page_context` object is a framework of a BSP and only plays a role in connection with the BSP extensions.

2 The `message` object is an attribute of the page object.

Table 3.5 Global Runtime Objects of a BSP Application

Table 3.6 provides a summary of which objects are available in which event handlers.

Event handler	Available global objects
onCreate	runtime, application, page_context, page (messages)
onRequest	runtime, application, page_context, page (messages), request, navigation, event_id
onInitialization	runtime, application, page_context, page (messages), request, response, navigation
onManipulation	runtime, application, page_context, page (messages), request, response
onInputProcessing	runtime, application, page_context, page (messages), request, navigation, event_id
onDestroy (not available)	runtime, application, page_context, page (messages)

Table 3.6 Global Runtime Objects in Event Handlers

The Processing Sequence

To start the application, enter the relevant URL into the address line of the Web browser. This address identifies the BSP application to be started. The start features of an application (BSP session) can be configured with various URL parameters, as already shown. Entering the URL results in a HTTP-GET-request that is sent to the BSO runtime. The BSP runtime then determines the appropriate BSP application and the requested BSP page. Depending on the setting, a logon to the SAP Web Application Server may be necessary.

Within the BSP runtime environment (BSP engine) the requested BSP page is processed, its components are passed through, and the corresponding processing steps are triggered according to the programmed logic. The result is a Web page that is sent to the initiator as a `response` object. The generation process takes place in different phases. In this process, events are run through in a specific sequence. The programmer has limited influence over this sequence. These events are represented by event handlers that carry out different tasks. The processing flow of a BSP application—in particular running through the various event handlers—is also affected by which status model is currently active. Figure 3.22 provides an overview of the schematic process flow (both for stateful as well as stateless).

When the page is called, the BSP runtime first determines whether or not there is already a page object for this BSP page. If this is not the case, the `onCreate` event handler is run. This generates the page object or an

Starting the
BSP application

Generation
and display

Processing flow
in the stateless
model

instance of it. From the developer's point of view, this handler can be used to initialize data or to generate required objects.

In the next step, any existing auto page attributes are transferred.

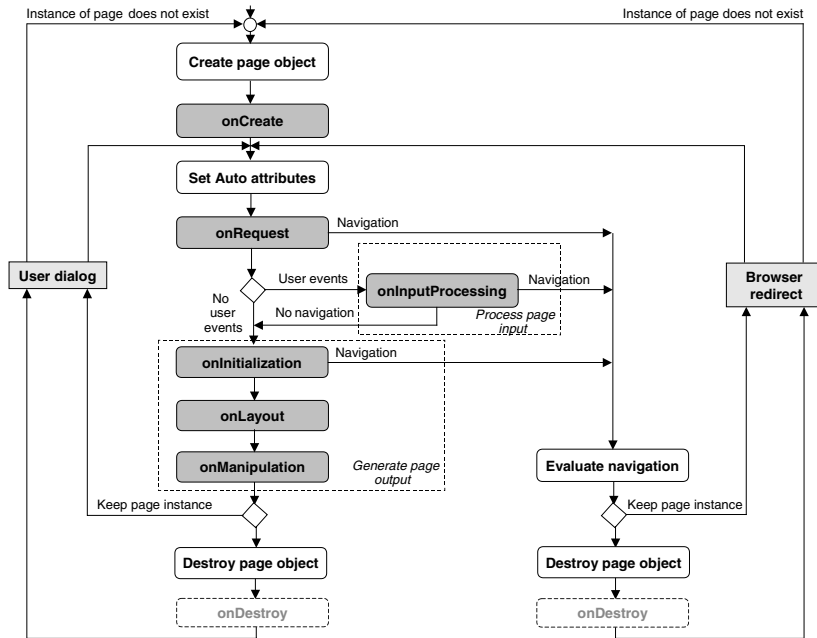


Figure 3.22 The Basic Processing Flow of a BSP Page

The `onRequest` event handler is then called. This handler is called on each request to a page. It is used to restore internal data structures from the request.

A decision now needs to be made on whether a user interaction has taken place. If this is the case, the `onInputProcessing` event handler is called; otherwise, the `onInitialization` event handler. Firstly we'll take a look at the situation without user interaction.

The `onInitialization` event handler is used to acquire the necessary data. This data may originate from many different kinds of sources (e.g. from DB tables, via function modules, BAPIs, etc.).

The `onLayout` phase is processed once the necessary data has been acquired. The design and the presentation of the page that the user sees are specified here. In other words, this involves the preparation and formatting of the requested pages. A page consists of static (e.g. HTML) and

dynamic (server-side scripting) parts. Whereas the client-side scripting (e.g. JavaScript) is returned unfiltered to the client, the server-side scripting (ABAP) is processed and transformed into coding that the browser can understand. The result is a serialized HTTP data flow.

One option for manipulating the HTTP data flow is the `onManipulation` event handler. This event handler is called after the `onLayout` event handler.

In the stateless case, the generated page object is destroyed again. In order to carry out any concluding work (e.g. saving the data in a server-side cookie), the `onDestroy` event handler is available.

This HTTP data flow is then sent to the initiator. The requested Web page then appears in the user's browser.

Only displaying the page however is not sufficient. The user should be able to input data, make selections or simply navigate to another page. So that this is possible, the relevant user interactions (e.g. via mouse and keyboard inputs) should be accepted and processed accordingly. The result could be a new Web page, the updated output page or even an error message, depending on the programmed logic. These user inputs are processed by the `onInputProcessing` event handler. If this type of user interaction has taken place, after `onRequest` the `onInputProcessing` event handler is called instead of `onInitialization`.

Response to user inputs and navigation

The `onInputProcessing` event handler is used to check incorrect inputs, the forwarding of attributes and the determining of follow-on pages for further navigation. If no follow-on page has been specified, the `onInitialization` event handler of the page is used to continue (navigation within the BSP). If, on the other hand, a follow-on page has been defined, the user will navigate automatically to this new BSP page. The follow-on page can be determined from the navigation structure or specified in the program code.¹⁵ The page requested is then created according to known processing flow and is output in the user's browser. A user action can now be made and the processing flow is re-started. With each navigation the underlying page object is destroyed again. In order to trigger the `onInputProcessing` event handler is triggered, a range of specifications needs to be observed during development. These are covered in Section 5.5.

¹⁵ Within this event handler, you can also determine a follow-on page dynamically by reviewing the usage details. You can find more information on this later in Chapter 5.

Processing flow in the stateful model

The processing flow is very similar to the stateless situation, so we will only cover the differences here. The most important feature is that the page object is retained¹⁶ for the entire duration of a session. When a BSP is first called, the `onCreate` event handler is called exactly once. If a navigation action is carried out within the BSP, the page object remains and `onCreate` will not be called again. The `onDestroy` event handler is only called if the system changes to stateless mode in the meantime. Otherwise processing takes place in the stateless situation.

- Exit** A stateful BSP application can be exited in two ways. The first option is to append special system parameters to the URL. The `sap-session-cmd=close` parameter ends the application. These special URL system parameters are described in section 3.3.2. With the second option, a timeout mechanism ends the session. If a BSP application is idle—that is, if no more actions are being carried out that result in another request/response cycle—a timeout intervenes after a time specified in the instance profile. The timeout also occurs if the Web browser is closed without the BSP application's being explicitly closed first. The SAP Web Application Server does not recognize when the Web browser is closed. This needs to be taken into account in the implementation.



A local ABAP class is generated from a BSP page during the runtime. The page layout and the various event handlers are methods of this class. The server scripting contained in a page is translated into the code of the layout method generated. The page attributes are also translated into parameter methods of the generated class.

3.3.4 Model View Controller Design Pattern

- Design pattern** In software development a *design pattern* is—in the broadest sense—a written document describing a general solution for a problem that crops up continually in a general form in many different types of project. Patterns provide a formal description of the problem, the solution and other factors. In object-oriented development a pattern of this type may contain descriptions of objects and classes including their individual components and dependencies. A collection of these patterns is called a pattern framework.

¹⁶ The life cycle of a page object can be specified on the page level, request level, or for the entire session.

The MVC design pattern describes a methodology to connect the userface with the underlying data model in an efficient manner. It is widely distributed in languages such as Java, Smalltalk, C and C++.¹⁷

The MVC design pattern contains a clear separation between the process flow, application logic (data model), and presentation logic. These three areas are formally divided using the three objects Model, View and Controller. This means that complex BSP applications can be divided into logical units. This has various advantages. Changes to the user interface have no effect on the application logic. Conversely, however, data can be presented multiple times in different display formats simultaneously. Clever update mechanisms change the data for an update in all displays.

Separation of process flow, application and interface

Interaction of the MVC Components

The components have already been introduced in brief in section 3.3.1, but are explained here once more to make things clearer.

► Model

The model represents the logical data structure of the data underlying the application. For example, it provides methods including backend services for data collection and processing. This component is usually responsible almost exclusively for implementing the application logic (business logic) and therefore does not contain any user interface information.

A model may have several Views that are realized using the relevant view pages.

► View

Views in typical applications consist of collections of classes for the representation of the graphical elements of the user interface, such as buttons, menus, and dialog boxes. Views enable the visualization of user interface elements.

In the SAP Web Application Server views are implemented as concrete characteristics of BSP pages. They contain HTML coding and ABAP for rendering the user interface.

To visualize the status, a view either sends queries to the model or the model informs the view of possible status changes. The view displayed to the client forwards actions by the user, such as clicking on a Submit button, to an assigned controller.

¹⁷ It is this that entitles the MVC model to call itself a design pattern. A basic requirement here is the level of distribution.



Views have neither event handlers nor auto page attributes. Page attributes are filled by the controller. Data binding permits attributes of the model classes that have been defined via page attributes, such as fields, structures and tables, to be called directly and also rewritten.

► Controllers

Controllers are the classes that establish the link between the Model and the View. They implement the decision-making process for responding to user inputs and control the process flow. Input data from the user accepted by the View is forwarded to the model here and triggers changes to the application data using the relevant method calls. The controller then causes Views to be executed, or changes the View's status.

The following points should be taken into account when using controllers:

- Controllers are either derived from the base class `CL_BSP_CONTROLLER2` (see Appendix A.3) or other controllers. This is easiest to achieve, as Figure 3.23 shows, via the forward navigation when creating a View.

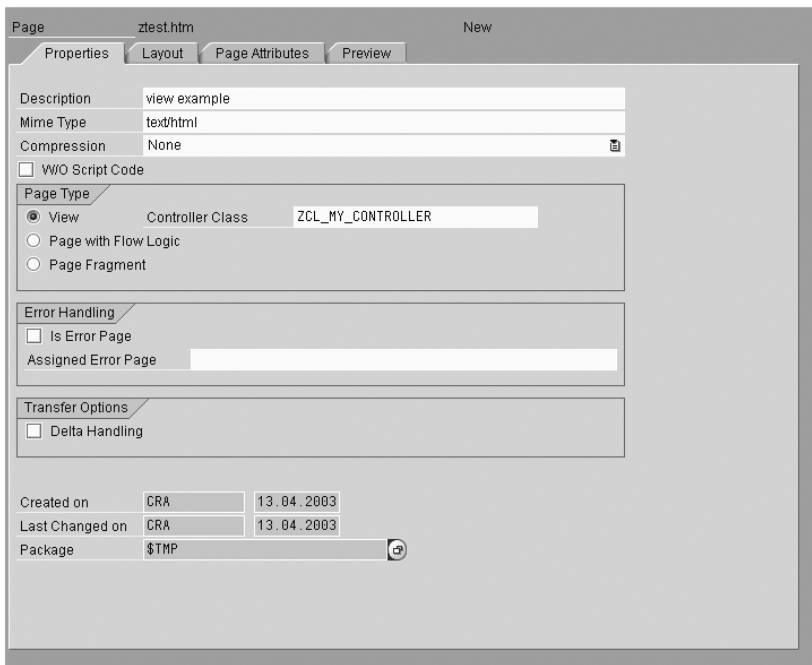


Figure 3.23 Creating a Controller via Forward Navigation

- ▶ Controllers can only control views from their own BSP application. However, the controls can be transferred to controllers from other applications.
- ▶ The life cycle of a controller is restricted to one call by default. By using the ID, the life cycle is taken from the settings of the controller properties. This facilitates a lifecycle `session` or a change of controller.
- ▶ Redirects are also possible using controllers.
- ▶ Controllers can be reached from outside via a URL. They possess the ending `.do`. A typical could also look like the following:

*http://www.my-webas.com/sap/bc/bsp/sap/zcode/
start.do?sap-client=100*

Figure 3.24 shows a schematic view of the connection between the Model, View and Controller components.

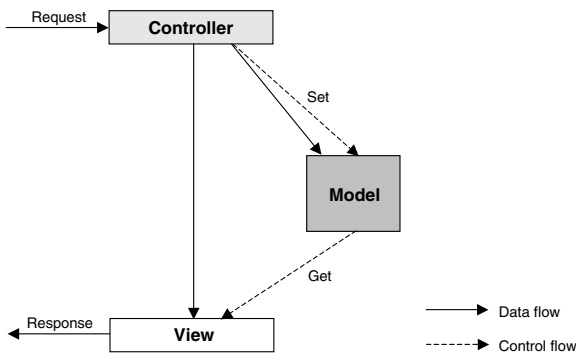


Figure 3.24 Schematic Interaction of the MVC Components

Implementation Examples

Figure 3.17 illustrates navigation in an example BSP application with flow logic. In a BSP application using the MVC Design Pattern, the display in Figure 3.25 would apply in the same way. Each of the BSP pages has implemented all three components. The Views are each called directly by their assigned Controller. It is possible to navigate between the BSP pages via redirect. It is also possible to assign several Models or Views to each Controller. Here you can see that "mixed mode" with typical BSP pages is also possible.

Simple
implementation

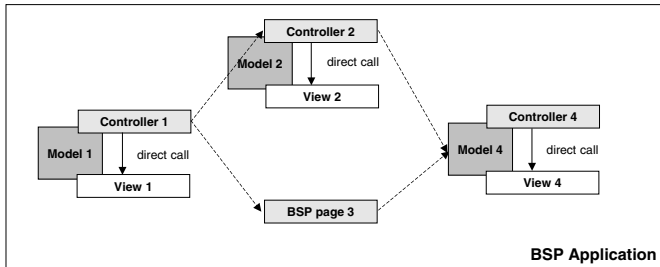


Figure 3.25 Simple BSP Application with MVC

Multiple views A Controller can control multiple Views in succession. With the corresponding flow control, it can also call these selectively. This is illustrated in Figure 3.26.

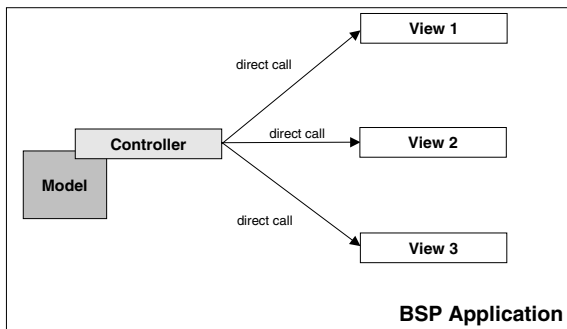


Figure 3.26 Multiple Views per Controller

Flexibility using central distribution

The flexibility is reflected in particular when combining the two previous options. In this process a superior central controller controls the distribution (this main controller does not require a separate view). Figure 3.27 illustrates this form of implementation.

It is also possible to generate a BSP page dynamically from multiple views. This form of componentization is not very easy to manage, however. Figure 3.28 illustrates this method of using controllers.

Naturally controllers from other applications can also be called. The examples show that there is a variety of combination options permitting virtually every conceivable implementation form. In addition, with all these combinations, BSP pages can be linked to the flow logic.

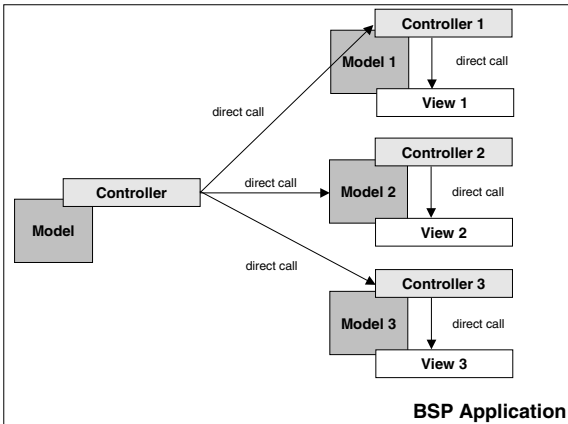


Figure 3.27 A Main Controller Controls the Central Distribution

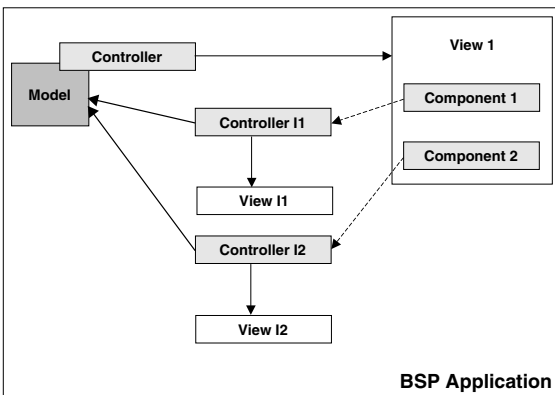


Figure 3.28 Making Components of BSP Pages with MVC

Combining MVC with Previous BSPs

The techniques for the BSP application programming model, when understood with the following rules, contribute to form the newly integrated MVC design pattern:

- ▶ Views can only be called by controllers. Exception: A call as an error page permits direct usage of a view.
- ▶ Controllers can call a controller using the `call` tag or the `goto` tag. However, they cannot call any pages using these tags.
- ▶ Transfers of pages to controllers and back may take place using `redirect` via the navigation methods.

In a BSP application, both pages with flow logic and controllers and views may be present.

Advantages of the MVC Design Pattern

The use of the MVC design pattern requires a certain amount of implementation work. This work is rewarded with the corresponding advantages:

► **Maintainability**

Thanks to the clean division of the presentation logic, flow control, and application logic, the structure is simplified. This means maintenance is simpler. Designers can focus on the design, and application designers on the application.

► **Performance**

The targeted use of the `goto` navigation reduces the number of required redirects and continues to use the same work process, which frequently saves resources.



As the MVC design pattern is only a concept, it is brought into "being" by conscientious implementation. A lack of distinct division eliminates any advantages of the concept. It is therefore advisable to reach a general consensus in development teams when faced with a difficult task for implementation.

Estimating the work required and the advantages

You should very carefully consider the increased amount of work required. The more complex a project or the more complex the requirements placed on a BSP application become, the more beneficial it is to use the MVC design pattern.

3.3.5 Web Dynpro

A major innovation came with the introduction of the *Web Dynpro* concept with SAP Web Application Server 6.40 for ABAP (see Figure 3.29). This generic concept permits the platform-independent creation and execution of modern Web-based interfaces by using graphic tools and programming.

The Web Dynpro model is characterized by a clear componentization, which ensures a high reusability of the individual components and simpler development and maintainence.

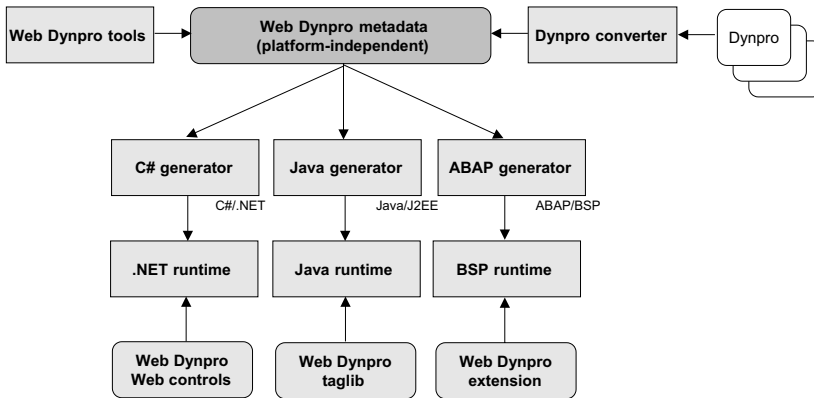


Figure 3.29 Web Dynpro Technology

The Web Dynpro technology offers the following advantages:

- ▶ Declarative and graphic tools make implementation easier
- ▶ Support of a structured design process
- ▶ Strict separation of layout data and business data (MVC)
- ▶ Reusability and better maintenance by using components
- ▶ Easily implemented layout changes and navigation in Web Dynpro tools
- ▶ Support of stateful applications
- ▶ Automatic data transport through data binding
- ▶ Automatic entry checks
- ▶ Syntax checks during development runtime

A simple concept ensures high general validity and platform independence of the interfaces: All elements of a Web application are generally written in XML. This XML description generates different rendering methods according to the runtime environment. The result is then transferred to the client. The main advantage of this principle lies in the independence of the runtime environment used and in the increased flexibility of connections to different clients.

To ensure this, the description of interface contents is based on metadata. From the model composed of the user interfaces, the associated code can be generated "at the touch of a button." This is the programming of previously defined events in the presentation logic requiring the addition of a separate application logic.

Metadata

Three different server code languages are supported. These include an ABAP code for the ABAP personality as well as Java code for the Java personality of the SAP Web Application Server and also C++ for the .NET environment. In addition, SAP provides a Dynpro converter that converts existing Dynpros (semi-) automatically into Web Dynpros. The common factor in the applications generated is that they are stateful, i.e. that session administration is a component of this concept. The user interface is modelled and developed using the MVC concept already described.

Until this point the server-side framework (SSF) for the Web Dynpro technology that runs on the SAP Web Application Server has been described. On the client-side, modern browsers can also use a Client-Side Framework (CSF) (see Figure 3.30).

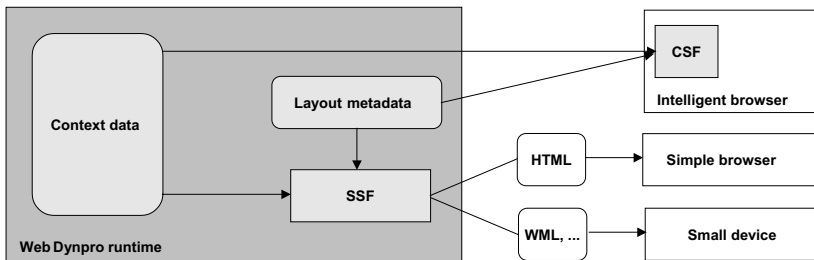


Figure 3.30 Server-Side- and Client-Side-Framework

This framework runs in the Web browser and is based on HTML templates, which are enhanced with JavaScript and Cascading Style Sheets. Web Dynpro checks in real-time if the client called (PDA, Desktop etc.) is compatible with this framework and sends the corresponding content as necessary.

The use of the client-side framework presents a variety of advantages. On the one hand the application becomes more user-friendly and has enhanced performance. The former is achieved on the one hand through concepts such as Drag & Drop, direct input checks such as field and value tools and on the other hand through delta handling. In this process, only data which have actually been changed for screen updates is transferred between the server and the client. This considerably reduces the bandwidth required.

The Web Dynpros development tool, which is called the Web Dynpro Explorer is completely integrated in the ABAP workbench.

The Web Dynpro Concept provides a range of predefined interface elements for creating Web interfaces. These various objects contain general user interactions as well as interface configurations. The various elements are grouped and compiled into libraries. They are independent from the frontend and the underlying application platform. The following libraries are available or predefined for the ABAP stack:

► **Standard**

This contains various interface elements to build the basic structure of a Web page. This includes buttons, input fields, tables, graphics, trees, scroll containers etc.

► **Office Integration**

This currently contains an element that allows you to link Excel and/or Word documents in a Microsoft view. This happens based on ActiveX-Controls.

► **Business Warehouse**

Reports are linked here.

► **Business Graphics**

These elements permit on the one hand graphic charts to be linked and on the other hand card contents to be displayed.

► **Adobe**

This provides an element, which links interactive PDF form on a Web page.

At the time of going to print the Web Dynpro concept had already been integrated into SAP Web Application Server 6.40, but was only released however for the J2EE Engine. We are presuming that Web Dynpro will also be released for future ABAP in future service packs. Due to further anticipated changes and expansions we will not discuss Web Dynpro in further detail in the practice chapter either.

3.4 Including Mobile Clients

As you have already learned, the client PC on which a Web browser is running is the central presentation layer in the foreground. At the same time the use of mobile terminals for business applications is becoming increasingly important. These mobile devices form the presentation layer for what is known as *mobile business*. Mobile business covers the non-location-specific procurement, processing, and provision of information of all kinds. Mobile business permits the processing of business and communication processes using mobile devices, suitable services, and net-

Index

.NET connector 44

A

ABAP 110, 579
 as a server-side script language 296
 global data declaration 113
 processing block 113
 Syntax 112
ABAP Dictionary 579
ABAP Editor 215, 579
ABAP interpreter 579
ABAP Objects 76, 111
ABAP Personality 43, 72, 202
ABAP Workbench 110, 209
 version management 229
Access areas
 public and private 415
Accessibility 548
Active caching 59
Addressing 175
 application name 176
 host 176
 name range 176
 page 176
 port 176
 protocol 175
 URL parameter 176
AGate 99
Alias
 external 247
 internal 245
Alias text 272
Apache 259
Application base class 174
Application class 172, 332, 339
 access 338
 assigning 338
 attribute creation 334
 creation 333
 stateful 173
 stateless 173
Application events 173
Application layer 31
 sub components 32
Application logic

 application class 172
 components 172
 model 175
Application name 176
Application platform 25
Application server 31, 579
Application toolbar 579
Audit Info System (AIS) 91
Authentication 490
Authorization 579
Authorization check 91
Automatic page attributes 322
 checking 359
 creation 324

B

Back-end system 579
BAPI 268, 579
BAPI Browser 268
 starting 269
Basic Authentication 92
Batch service 38
BEE 439
 calling 440
 creating with exactly one BSP
 element 448
 creating with HTML code 446
 creating with XML coding 449
 creation with several BSP elements
 451
 factory method 448
BEE hives 440, 445
Benchmark extension 509
BOR 579
Browser session 184
BSP application 172, 283
 access 175
 access areas 415
 addressing 175
 and OTR texts 271
 assigning a package 285
 assignment of a theme 236
 components 165
 creation 284
 definition 164

- dynamic navigation 320
- error messages 177
- event handler-controlled processing
 - 182
- event-control 318
- exit 194
- flow control 171
- generation and display 191
- global runtime objects 190
- HTML form control 326
- layout design 254, 263
- layout structure 257
- management attributes 166
- MIME objects 293
- presentation components 166
- request handler 466
- setting the start page 292
- starting 191
- stateful and stateless 184, 369
- status model 183
- BSP directives 295, 574
- BSP element 510
 - BEE 439
 - breadCrumb 510
 - button 384, 510
 - chart 510
 - checkbox 511
 - composite 407
 - content 382
 - dateNavigator 511
 - document 383
 - dropDownListBox 512
 - event handling 394, 396
 - fileUpload 512
 - form 383
 - generation 403
 - gridLayout 390, 512
 - gridLayoutCell 391
 - group 513
 - inputField 513
 - itemList 513
 - listBox 514
 - modifying 398
 - page 383
 - radioButtonGroup 514
 - tableColumn 386
 - tableView 385, 514
 - tabStrip 514
 - tray 515
 - tree 516
 - use 382
- BSP extensions 77, 154, 169, 381, 509
 - advantages 170
 - creation 402
 - directive 382
 - HTMLB 381
- BSP page 166
 - creation 288
 - data definition 303
 - event handler 303
 - layout 303
 - layout structure 290
 - page attributes 167, 305
 - page fragment 167
 - parameter transfer 322
 - preview function 292
 - server-side scripting 295
 - testing 291
 - type definition 167, 303
 - with flow logic 166
- BSP session 183
- BTF extension 509
- Buffer 579
- Business Connector 579
- Business framework 579
- Business intelligence 23
- Business layer 33
- Business object 579
- Business Object Repository 268, 579
- Business Process Management 25
- Business Server Pages 41, 44

C

- C# 202
- Caching 36, 57
- Calendar data 351
- Central Services 68
- Chain rule 113
- Change request 579
- Changing the language version 347
- Changing the system language 347
- Class Builder 216, 579
- Class model 339
- Classes
 - CL_BSP_CONTROLLER 556
 - CL_BSP_GET_TEXT_BY_ALIAS 555

- CL_BSP_MESSAGES 553
- CL_BSP_MIMES 560
- CL_BSP_SERVER_SIDE_COOKIE 558
- Client 579
- Client Layer 31
- Client Server Principle 31
- Client-side JavaScript 110
- Collaboration 22
- Command field 580
- Comments 300
- Common Logfile Format 576
- Communication interface 580
- Composite Application Framework 26
- Composite elements 407
- Connectivity layer 32
- Constructor 333
- Controller class 423
- Controller object 423
- Controllers 77, 171, 196, 442
 - creation 423
 - output processing 427
- Cookies 160, 527
 - client-side 161, 373
 - non-persistent 161
 - persistent 160
 - server-side 163, 376, 397
- CPI-C 39
- CSS 148, 300, 580
 - advantages 148
 - inclusion 149
 - options 149
 - versions 149
- Customizing 307, 580

D

- Data definitions 303
- Data dictionary 306, 580
- Data element 580
- Data model 195
- Data retrieval 302
- Database interface 580
- Database layer 33
- Database server 580
- DDIC Services 350
- Debugger 580
- Debugging 310
- Default user 418

- Deserialization 280
- Design pattern, definition 194
- Developer key 580
- Developing Web services 86
- Development database 229
- Device detection 205
- DHTML 114
 - Browser incompatibility 153
- DIAG 580
- Dialog service 38
- Dictionary services 350
- Digital certificate 580
- Dispatcher Queue 39
- DMZ 89
- Document Object Model (DOM) 154
- Domain 580
- Drag & Drop 580
- Drag & Relate 580
- Dreamweaver 259
- DTD 82, 117
- Dynamic caching 59
- Dynamic HTML 114
- Dynpro 580

E

- Eclipse 46
- Eclipse Framework 78
- E-mails 42
- Enqueue service 38, 69
- Enterprise JavaBeans 79
- Enterprise Services Architecture 20
- Entity 581
- Entry checking 358
- Error messages 177
- Escaped-URL 176
- Event control 318
- Event handler 144, 171, 183, 189, 303, 306
 - global runtime objects 191
 - implementation 307
 - onCreate 189
 - onDestroy 190
 - onInitialization 189
 - onInputProcessing 190
 - onLayout 189
 - onManipulation 189
 - onRequest 189
 - processing sequence 191

- Event ID 319
- Extension directive 170, 171
- External alias 247

F

- Factory method 440
- Fault message 505
- Fault tolerance 35
- Field description 350
- File Access Handler 53
- Firewall 581
- Flight model 106, 107, 108
- Flow control 171
- Formatting options 341
- Formatting the output 339
 - WRITE 340
- Forward navigation 581
- Frames 119, 581
- Function builder 581
- Function module 581

G

- Gateway 581
- Gateway service 38
- Generic Security Services (GSS-API) 91
- GoLive 259, 263, 264
- Graphics extension 509

H

- Help values 353
- Hidden form fields 372
- HTML 110, 114, 115, 581
 - history 119
 - mark-up language 115
 - static 165
- HTML automatism 329
- HTML business 170
- HTML file 118
 - heading section 118
- HTML form control 326
- HTML form fields 526
- HTMLB extensions 170, 381, 509
- HTTP 154, 155, 581
 - body data 528
 - breakpoints 250
 - debugging 216, 249
 - framework 60
 - header 525

- multipart data 529
- plug-in 52
- request handler 62
- server 581
 - Service hierarchy 243

- HTTP-NG 155
- HTTPS 156, 581
- Hyperlinks 116

I

- IBM 78
- ICF 60, 67, 93, 516
 - class model 516
 - Client role 65
 - details 479
 - Server Monitor 254
 - Server role 63
 - service hierarchy 239
 - services 91
- ICM 39, 41, 42
 - architecture 48
 - HTTP plug-in 52
 - Logging 574
- ICM Server Cache 56, 531
- ICM Server Clipboard 59
- IF_BSP_APPLICATION interfaces 542
- IF_BSP_PAGE interfaces 549
- IF_BSP_PAGE_CONTEXT interfaces 551
- IF_BSP_RUNTIME interfaces 547
- IF_CLIENT_INFO interfaces 560
- IF_HTTP_CLIENT interfaces 521
- IF_HTTP_ENTITY interfaces 524
- IF_HTTP_EXTENSION interfaces 516
- IF_HTTP_HEADER_FIELDS_SAP interfaces 536
- IF_HTTP_SERVER interfaces 518
- IF_HTTP_STATUS interfaces 532
- Inbound calls 74
- Include directive 167
- Info system authorizations 91
- Information integration 23
- Inline code 295
- Instance 581
- Integration layer 33
- Interface repository 581
- Interfaces
 - IF_BSP_APPLICATION_EVENTS 543

- IF_BSP_NAVIGATION 544
- IF_BSP_SERVICES 552
- IF_HTTP_FORM_FIELDS_SAP 540
- IF_HTTP_HEADER_FIELDS 534
- IF_HTTP_PROXY_CONFIG 541
- IF_HTTP_REQUEST 529
- IF_HTTP_RESPONSE 530
- IF_HTTP_UTILITY 531
- Internal alias 245, 347
- Internet Application Components 99
- Internet Communication Framework 60
- Internet Communication Manager 39, 48
- Internet Transaction Server (ITS) 98, 581

J

- J2EE application server 68
- J2EE architecture 68, 79
- J2EE engine 72
- J2EE programming interfaces 70
- J2EE Startup and Control Framework 68
- JAAS 72
- Java connector 44
- Java engine 46
- Java instance 68
- Java Mail 71
- Java persistence 69
- Java Personality 43, 73
- Java Server Pages 45, 70, 79
- Java Servlets 79
- JavaBeans 70
- JavaScript 110, 121
 - checking the date 364
 - clientside 363
 - conditions 122
 - data types 127
 - date check 367
 - frames 145
 - functions 129
 - literals/constants 128
 - loops 125
 - methods 135
 - objects 135, 137
 - operators 124
 - variable declaration 127

- variables 126
- JCA 71
- JCE 71
- JDBC 71
- JMS 71
- JNDI 71
- JTA 71
- JTS 71

K

- Knowledge Management 23

L

- LAN 581
- Layout 303
- LDAP 91
- Lifecycle Management 27
- Linking MIME objects 294
- Load balancing 35, 96
- Logging handler 53, 54
- Logical ports 498
 - configuration 500
- Logon procedure 91

M

- Mass import, MIMEs 233
- Master data 581
- Master Data Management 24
- Memory Cache 58
- Memory pipes 50
- Message object 358
 - entry check 361
 - error handling 363
 - processing user entries 360
- Message service 39, 68
- Metadata 582
- Microsoft IIS 259
- Middle layer 31
- MIME objects 164, 168, 231, 256, 257
 - CSS 300
 - HTTP Service hierarchy 241
 - importing 293
 - language versions 236
 - mass import 233, 262
 - themes 235
- MIME Repository 231, 257, 262
- MIME types 572
- MOB_DEVCFG, table 205, 206

- Mobile business 203
 - Mobile clients 203
 - Mobile terminals 204
 - Mode 582
 - Model 77, 175, 195
 - Model class
 - creating 431
 - Modification 582
 - Multilingual capabilities 344
 - Multiparts 529
 - MVC 77, 166, 171, 195, 422
 - advantages 200
 - combination with previous BSPs 199
 - controllers 196
 - implementation examples 197
 - interaction of the components 195
 - making components 198
 - model 195
 - view 195
 - mySAP Business Suite 582
- N**
- Name range 176
 - Name/Value Pairs 326
 - Native SQL 582
 - Navigation 318, 322
 - Navigation request 171, 321
 - Navigation structure 171
 - Network Address Translators (NATs) 96
- O**
- Object Navigator 76, 209
 - browser area 211
 - configuration 215
 - function keys 213
 - navigation stack 213
 - object list 212
 - Repository Browser 218
 - Repository Infosystem 222
 - status bar 212
 - toolbars 210
 - tools area 212
 - Transport Organizer 223
 - worklist 214
 - Object navigator 582
 - onCreate 189
 - onDestroy 190
 - onInitialization 189
 - onInputProcessing 190
 - onLayout 189
 - Online Text Repository (OTR) 271, 344
 - onManipulation 189
 - onRequest 189
 - Open SQL 39, 582
 - Open SQL for Java 70
 - OTR browser 273
 - OTR concept 274
 - OTR long texts 274
 - OTR short texts 272, 348
 - OTR transport 275
 - Outbound calls 74
 - Output formatting
 - special options 343
 - TO_STRING 340
- P**
- Package 219, 582
 - Page attributes 305, 355, 356, 429
 - creation 309
 - Page declarations 382
 - Page fragments 167, 298, 301
 - creation 298
 - filling 300
 - include 300
 - PAGE object 339
 - Pages with flow logic 290
 - People integration 22
 - Persistence layer 33
 - Personalization 303
 - Ping service 242
 - Platform 582
 - Plug-in 582
 - Portability 582
 - Portal 582
 - Presentation layer 32
 - Pretty Printer 216
 - Print directive 296
 - Process integration 24
 - Processing flow 426
 - Profile generator 91
 - Protected services 418
 - Protocol 582
 - Proxy generation 496
 - Pseudo header 526

Public key infrastructure (PKI) 91
Public services 417
Publishing 84

Q

Query string 179

R

R/3 database 39
R/3 Webservice Repository 87
RDBMS 582
Redirect Handler 53
Register 582
Remote Function Call 39, 268, 582
Remote system 582
Repository 582
Repository Browser 218
 MIME object lists 233
Repository Infosystem 222
Request handler 466
Return on investment 20
Reverse proxies 96
RFC-enabled Function Module 268
RMI IIOP 71
Roll area 582
Runtime Analysis 252
Run-time Environment 583

S

SAP basis 583
SAP Data Modeler 106
SAP DCOM 583
SAP easy access 583
SAP Enterprise Portal 22, 584
SAP Exchange Infrastructure 24
SAP GUI 37, 583
SAP Help Portal 583
SAP IGS 509
SAP Internet Graphics Server 509
SAP J2EE Engine 46, 68
SAP J2EE Engine characteristics 70
SAP Java Connector (JCo) 46
SAP Library 583
SAP Logon Tickets 91
SAP NetWeaver 19
 security aspects 28
SAP NetWeaver Developer Studio 78
SAP R/3 31, 37

SAP R/3 Handler 54
SAP Service Marketplace 583
sap sessioncmd 180
SAP Solution Manager 27
SAP Web Application Server 205
 applications 47
 architecture 37
 as client 42, 470
 characteristics 44
 definition 30
 J2EE conformity 70
 J2EE Engine 68
 Java support 40, 46
 logon 291
 MIME objects 231
 security 45
 supported MIME types 572
 XML support 72
SAP Web Dispatcher 73, 96, 97
 Characteristics 97
sap-client 181
sap-exiturl 180
sap-password 182
SAProuter 91
sap-syscmd 181
sap-user 181
Screen Painter 422
Security Audit Log 91
Security mechanisms 28
Serialization 280
Server Cache Handler 53, 54
Server-side Scripting 110, 295
Service maintenance 238
 HTTP debugging 249
 runtime analysis 252
 services 238
 trace 250
Service options 94
Services 238
 activation and deactivation 242
 creating 239
 defining properties 240
 Handler List 241
Session cookies 184
Session handling 583
Session ID 184
Session-oriented communication 490
SGML 114, 148, 157

- S-HTTP 156
- Signal handler 50
- Simple Mail Transfer Protocol 54
- Simple Transformation 278
- Single Sign-On (SSO) 91, 583
- SMTP Client 55
- SMTP plug-in 54
- SMTP Request Handler 66
- SMTP Server 55
- SOAP 80
- SOAP framework 44
- SOAP request 81
- SOAP response 81
- Software Deployment Manager 40, 69
- Spool service 38
- SQL 584
- SSL 156, 584
- Standard BSP Handler 62
- Standards 44
- State model 369
- Stateful mode 40
- Stateful model 184, 194
- Stateful principle 64
- Stateless mode 40
- Stateless model 186, 191
- Stateless principle 64
- Stateless Web Applications 110
- Status bar 584
- Status models 183
- Subhandlers 52
- Syntax beautifier 227
- Syntax check 296
- System function bar 584
- System landscape 584
- System logon 178
- System variables 113

T

- tableView Iterator 457
- Tag Browser 234
- Tags 116
- TCP/IP 154, 584
- Template 584
- template function 338
- Themes 168
- Themes editor 168, 234
- Thread Control 49
- Three-tier architecture 31

- Ticket 584
- tModels 87
- TMS 584
- Tool tip 351, 356
- Trace 250
- Trace file 251
- Transaction 584
- Transaction code 584
- Transaction data 584
- Transformation Editor 275, 276
 - functionalities 276
- Transport Guarantee 491
- Transport management system 584
- Transport organizer 223, 584
- Transport request 286, 579
- Type definitions 303, 313

U

- UDDI 80, 84
- UDDI registry 84
- UFO Cache 58
- Update service 38
- URL 584
 - parsing 249
- URL parameter 176, 179, 325
 - system-specific 180
- URL-mangling code 182
- User context 584
- User entries 317
- User menu 584

V

- Version database 229
- Version management 229
- View 77, 167, 195, 584
 - calling 429
 - creation 428
- Virtual hosts 244
 - creating 244
 - setting properties 245
- Virtual interface 86
 - creation 473

W

- WAP 204, 585
- Watchdog 50
- Web Application Builder 209, 223, 585
 - Editor 224

- MIME Repository 231
- Tag Browser 234
- Theme Editor 234
- Version management 229
- Web Application Server 18
 - architecture 31
 - demands 34
 - development environment 34
 - scalability 35
 - security 36
 - transaction security 36
- Web Dynpro 77, 200, 203, 422
- Web folder 259
 - creating 260
 - naming 261
- Web Service Creation Wizard 86, 473, 492
- Web services 79, 84, 473
 - architecture 85
 - basic principles 80
 - browser 87
 - calling 494
 - configuration 474
 - creation 474
 - error handling 504
 - framework 85
 - homepage 87, 484
 - implementation 501
 - security 479, 481
- WebDAV 255, 259
 - access control 255
 - advanced collections 255
 - collections 255
 - locking 255
 - propfind 255
 - settings 266
- WebDAV client
 - lock 258
- WebDAV interface 254
 - MIME import 233
- WebDAV server 265, 267
- WebDAV service 256
- WGate 99
- WML 204
- Work process 585
- Workbench 585
- Worker thread 49
- WRITE, method 340
- WSDL 80, 82
- WSDL Generation 483

X

- X.509 client certificates 91, 92
- XHTML 114
- XML 80, 156, 585
 - History 160
- XML code
 - creation 450
- XML Documents
 - transformation 275
- XML Interpreter 158
- XML Schema 82
- XSL 149
- XSLT 278
- XSLT processor 277