Index

- What is the Protal Development Kit (PDK)

- **Setting Up**
  - System requirements and necessary tools
  - Installation of the Portal Development Kit (PDK)
  - Installation of the Knowledge Management (KM)
  - Administration of the PDK
  - **Setting up a development enviroment for the PDK**
    - Installation of the PDK Wizards for Borland JBuilder - Plugins for the IDE

      Setting up JBuilder for efficient collaboration with the Portal Development Kit.
    - Tomcat settings for remote debugging with JBuilder
    - **Working without wizards or other IDE**
      - Setting up a JBuilder Project

        As an example we will explain how to build the BasicExample project.
      - Ant as a Building Tool

        Create par files using the Ant tool, using the custom Ant task 'deploy'

- **Creating a Portal Component**
  - Introduction to the Portal Component Development

    includes: what is a component; Par File structure;
  - Development Guide using the PDK Wizards

    Portal Archive Development Cycle Using JBuilder; Users Guide for the PDK Wizards
  - Libraries for the PDK

    Library names and location in the TOMCAT_home folder.
  - Portal Component Structure for Correct Deployment

    Describes the recommended par file structure. The par file structure is necessary to make sure that the files end up in the right place on the servlet engine after the deployment.
  - Example displays a text on the web client using a DynPage

    This example will show the text Hello World . The user interface is implemented with HTML Business for Java extending the class DynPage

- **Adding Services**
  - Introduction to Services and Profiles
  - Example displays a text on the web client using a DynPage. The text is read

from the properties file

The second DynPage example shows you how to read data from the properties file.

- **Example displays a text on the web client and expects user input using a DynPage. The text is read from the properties file and the user input is saved in the properties file**

  This example uses a grid layout for better positioning of the controls, input fields and buttons. The displayed text is read from the properties file. The user input is saved in the properties file.

- **Tutorials**
  - **Building a Portal Component with HTMLB using a JSP DynPage**

    DynPage;Basic Example;Event Handling;Using a bean;Error Messages
  - **Building a Portal Component with Graphical User Interface and Event Handling**

    Creating a portal component with a graphical user interface and event handling. Accessing the personalization data (properties). Accessing resources. Using a portal service.
  - **Internationalization of the Portal Component: Portaloto**

    Creating resource bundles for English and German language. Changing JSP's and DynPage to access the resource bundles.

- **Common Problems during Installation or Development and Their Solutions**
  - **Frequently Asked Questions**

What is the Portal Development Kit (PDK)

Introduction

Portals enable the web browser to access information, services and applications. Basic portal features are:

- Build iViews with portal components which show content.
- Allow users to personalize the content of portal component as well as their visual appearance in the pages, provide ways to navigate between pages and external services, personalize the navigation etc..
- Basic knowledge/content management features such as search, document management, teamwork support.
- Administration features for the portal, its users (permissions to access content), change the GUI (Graphical User Interface), etc.

to name a few.

An iView contains information and functions from all kind of sources. An iView can present information from a Web site, integrate functions from business software, provide search functions and so on.
iViews are usually programmed in Java or use Java Server Pages (JSP) technology or are a combination of Java and JSP and run on the iView Runtime servlet engine (e.g. Tomcat). To develop iViews you can use the Application Programming Interface (API) of the Java iView Runtime.

Portal Component

A portal component consists out of one or more master iviews. A portal component has an application part (e.g. a Java class and/or Java Server Pages (JSP)) and can use resources (e.g. images, language bundles etc.). A portal component needs a property file that tells the portal runtime which class to use when the portal component is called, which services the portal component needs, in which mode the portal component should run - to name a few. All the files that belong to a portal component are packaged into a par file (PAR = Portal Archive). The par file is than deployed into the PDK local portal or finally into the Enterprise Portal (EP). The par file uses the zip format and can be accessed with all programs capable reading zip files.

The PDK provides two methods to create a portal component.

- Abstract Portal Component
  Components using the Abstract Portal Component class offer a lean method to write HTML commands to the web client. The Abstract Portal Component offers basic event handling, so that for larger, interactive components the programming effort is higher.
  Abstract Portal Component example.

- DynPage
  The Page Processor Component returns a DynPage. It provides a more sophisticated event handling and easier access of JSP (Java Server Pages) files.
  DynPage example.

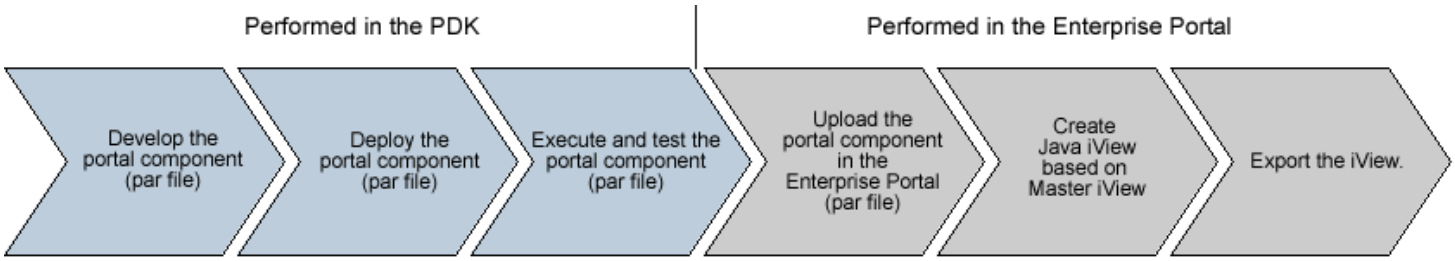
Note:
The PDK documentation and examples focus on the DynPage. The easier event handling and the idea to separate the content development (JSP) from the application development (Java) make the DynPage a better choice for components with interaction and changing content.

Portal Development Kit (PDK)

The PDK provides the environment necessary to create and execute portal components and services locally. The portal components can be tested locally before finally deployed in the Enterprise Portal (EP). The PDK requires a local servlet engine (e.g. Tomcat) and a Java development tool (e.g. JBuilder). The "Getting Started" section of the PDK documentation gives you all the necessary information to get your development started.

Developing Java iViews

The typical steps to develop a Java iView are.



Development, execution and test of portal components can take place in the PDK. A final test in the Enterprise Portal (EP) environment is necessary.

Differences between the Enterprise Portal (EP) and the PDK.

Enterprise Portal (EP)	PDK
iFrame Rendering	Table Rendering (Page Builder). To emulate the EP use property `Isolated` in PDK page builder.
Isolation Mode (iView Server)	not available
Servlet Engine: SAP J2EE or JRun	Servlet Engine: Tomcat No Enterprise Java Bean (EJB) support.
LDAP	Text file.

How can the PDK Help?

The PDK is a collection of information and tools for a developer.

Developer support:

- Developer's Portal - for Portal Content Developers.
- Documentation.
- Examples.
- All required Java libraries (jar files).
- Wizards to create portal archive (.par) files.
  Wizards are plugins for IDE's like Borland JBuilder.
- Test container to run/debug and test portal components locally.
- Platform service API's.

Offered services:

- HTML Business for Java (HTMLB).
  Generate browser independent HTML pages in SAP Style. HTMLB provides powerful controls like chart, table view, tree etc. to develop sophisticated and interactive graphical user interfaces.
- Internationalization.
  Access the required language resource files for multilingual applications.
- Portal Data Viewer (PDV).
  Present data from different sources (XML, JDBC query etc.) in tabular form.
- Logger.
  Log events for diagnosis or statistics.
- JCo.
  JCo connects you to SAP systems.
- User Management.
  Retrieve User Mapping information.
- Enterprise Portal Client Framework (EPCF)
  Eventing between iViews on the web client (browser).

Skills necessary for the PDK

Basic knowledge of :

- Java.
- Basic operating system skills (Navigation and file copy).

- JSP (Java Server Pages) skills are helpful.
- HTML skills are helpful.

About the documentation

The PDK documentation assumes that the local portal is installed and up and running so that the examples can be started directly from the documentation. If you have no portal you can read the documentation but links which start an example will produce an error message (like page not found).

Prerequisites

To use the PDK (Portal Development Kit) your PC must meet following requirements:

Hardware Requirements

Minimum requirements:

400 MHz CPU
256 MB RAM
600 MB free Diskspace

Operating System

Windows NT
Windows 2000

Software Requirements

Java Development Kit

You have to install the Java Development Kit Version 1.3.1. You can download this version from the Sun - Jave homepage.

Sun: http://java.sun.com/j2se/

Development Environment

The PDK documentation assumes that you work with Borland's JBuilder 5. You can use other Java development tools but you have to adjust the step by step instructions for JBuilder appropriate to your development tool.

Borland: http://www.borland.com/jbuilder/

Servlet Engine

The PDK is build and tested on the Tomcat servlet engine. In case you use any other J2EE servlet engine we can offer no support.

The PDK will use Tomcat Version 3.3 or latest patches 3.3.x. Please install Tomcat on your PC. To install Tomcat download the installation file (zip format) from

> Jakarta Project: http://jakarta.apache.org/tomcat/

Important:

Don't forget to set the environment variable TOMCAT_HOME according your Tomcat installation.

The JSP engine also uses 'javac' therefor the tools.jar file needs to be in the CLASSPATH. To do that you have 3 options:

- Set JAVA_HOME to the directory where JDK is installed.
- Put "tools.jar" in your CLASSPATH.
- Copy "tools.jar" into the Tomcat lib/apps folder.

Build Tool

The Apache Ant build tool is optional.

> Jakarta Project: http://jakarta.apache.org/ant/

ZIP Tool

We recommend a programm (Winzip or "compatible") capable to create and read compressed files in ZIP, JAR and PAR format.

Installation on Windows NT/2000

In this chapter you will install the Portal Development Kit on your local PC. Before you can start with the installation you have to check if the hardware and software requirements fits to your PC.

If you have already an older version of the PDK installed you have to physically remove (do **not** just rename the folder !) the folder `%TOMCAT_HOME%/webapps/irj` before you install the new PDK.

Note: All changes you have made in the PDK are lost when you remove the irj folder!!! If you want to save your work you could move the `%TOMCAT_HOME%/webapps/irj` folder to e.g. C:\TEMP\irj.

1. Make sure that the environment variable `%TOMCAT_HOME%` is set to your tomcat installation.
2. **Download** the Portal Development Kit from the iView Studio .
3. **Extract** the zip file pdk_tomcat.zip in your tomcat root folder (e.g. c:\tomcat)..
4. Modify the `tomcat.bat` file in `%TOMCAT_HOME%\bin`. You must set the proxy settings in line 48 (_PROXY_HOST) and 49 (_PROXY_PORT) according your Internet browser settings.
5. **Start Tomcat** (execute c:\tomcat\bin**startup.bat**).
   This will automatically deploy all portal components and portal services.
   You will get the following "error" message in the dos console:

   ```
   Jul 31, 2002 7:54:21 PM # IRJ Init thread      Fatal           LOCK_CLIENT - OK: Lock
   client connected.
   Server: xxx client: xxx port: 3299 pid: 1 #
   ```

   This is not an error message but a success message. But the message seams to have the wrong log level;-)
6. Start your local portal with http://localhost:8080/irj/servlet/prt/portal

When you logon the first time after starting tomcat you will get a popup with a warning, that you have not got a valid license.

You can ignore this message. The Portal Development Kit comes as a test container. You can use it with up to 2 named user. If you want to use it with more users or avoid the warning popup after restarting tomcat you have to install a license key.

The procedure on how to get a license key is described in the standard documentation of EP 5.0.

Installation of JBuilder open tools

1. Start Tomcat.
2. Start your local portal with http://localhost:8080/irj/servlet/prt/portal
3. Navigate to `{6} Downloads -> {2} IDE Plug-In .`

Using Knowledge Management

If you want to use the Knowledge Management you must continue with Installing and Configuring the Knowledge Management

Before you can start with this installation you must install the Portal Platform.

Installing and Configuring the Knowledge Management

Install the Microsoft SQL Server

You can download a evaluation copy from
http://www.microsoft.com/sql/evaluation/trial/2000/download.asp

1. Start the installation.
2. The SQL authentication method must be configured to Mixed Mode (Windows Authentication and SQL Server Authentication).

Create the databases for KM in the SQL Server

1. Start the SQL Query Analyser.
2. Open the File WCM.sql from ... (see dbpatch.zip within your PDK installation)
3. Select the database **master** in the Dropdown List in the Standard Button bar.
4. Execute the Query.
   You created the databases wcm and wcm_acl.

Now you can use the Knowledge Management iViews.

Administration of the PDK

Create your own User

The user management of the PDK uses the local text file `KMUsers.properties` to store the user information. The file is located in `<TOMCAT_HOME>\webapps\irj\Web-inf\plugins\portal\services\usermanagement\`.
To create a new user or modify the default user do:

- Open KMUsers.properties in `<TOMCAT_HOME>\webapps\irj\Web-inf`\plugins\portal\services\usermanagement\data
  - Copy the User wp entries
  - Change the id of the new user from wp to yourname (change wp.* to yourname.*)
- Restart tomcat
- Start the portal in the browser with the address http://localhost:8080/irj/servlet/prt/portal

HINT: If you want to use the Single Sign On functionality with SAP systems the User-Id must be identical to the SAP user you want to login with.

Setting up your System Landscape

There are two files that define the System Landscape:

- `systems.xml` in the folder `<TOMCAT_HOME>\webapps\irj\Web-inf\plug-ins\portal\services\landscape\xml\`
- `jcoDestination.xml` in the folder `in the browser with the address\webapps\irj\Web-inf\plug-ins\portal\system\xml\`

A list of all attributes and how to maintain the files can be found in the standard Enterprise Portal documentation `Help for the Portal Administrator` in the chapter `System Landscape`. Find the documentation at http://help.sap.com/portals/.

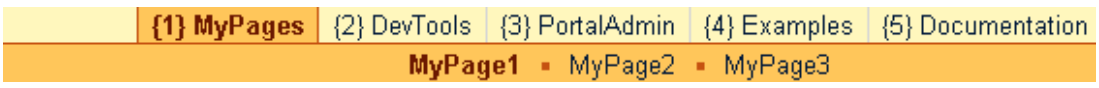
Setting up your portal for Single Sign-On

Information is available in the documentation `Help for the Portal Administrator` in the chapter `Security -> Single Sign-On`. Find the documentation at http://help.sap.com/portals/.

Defining the Navigation Structure of your Portal

The Top Level Navigation is defined in the folder `<TOMCAT_HOME>\webapps\irj\Web-inf`\plugins\portal\content. Every sub folder is displayed in the First Line of the Top Level Navigation.

Example: The content folder contains the sub folders **{1} My Pages**, **{2} DevTools**, **{3} PortalAdmin**, **{4} Examples** and **{5} Documentation.**
Result :



Hint: The {1}..., {2}... format is used to force the file system to place the sub folders in ascending order.

The sub folders contain the page definition files in XML format. According to the example above the sub folder **{1} My Pages** contains the page definition files **MyPage1.xml** , **MyPage2.xml** and **MyPage3.xml**.

Create your own Page

To create pages you can use the PageEditor. In the PDK you can find it at {2} DevTools -> {3} PageEditor.
To create a page manually you must do the following:

- Open folder `<TOMCAT_HOME>\webapps\irj\Web-inf\plugins\portal\content` in the Windows Explorer
- If you want a new entry in the first line of the Top Level Navigation create a new sub folder e.g. `myfolder.`
- Copy an appropriate template for your component (full size, narrow and so on) from `<TOMCAT_HOME>\webapps\irj\Web-inf\deployment\Pcd\global\pages\templates-1024x768` (or 800x600 according to the resolution you want to use on your web client) to the sub folder you just created, `<TOMCAT_HOME>\webapps\irj\Web-inf\plugins\portal\content\myfolder`, and rename it to `mypage.xml`
- Edit the `mypage.xml` file to set title, width, color and so on. (See example below)
- Start the portal in the browser with the address http://localhost:8080/irj/servlet/prt/portal. You should see the new tab with the new page in your portal.

Example

```
<page>
        <container title="Workspace"
                           id="Workspace"
                           preferredSize="FullSize"
                           direction="vertical"
                           width="100%">
                  <component name="StyleSheetDesigner.default"
                                    title="StyleSheetDesigner.default"
                                    trayType="SAPTrayD3"
                                    isolated="true"
                                    height="800px"/>
        </container>
</page>
```

Page Definition in Detail

All portal components are place in a container. You have the choice between a horizontal or a vertical container. The container is defined in the page description XML file as follows:

```
<container direction="vertical" width="100%"> </container>
or
<container direction="horizontal" width="100%"> </container>
```

To place your own portal component in the container you have to specify the name of your portal component (ComponentName.PropertyFile) in the `component` tag:

```
<component name="StyleSheetDesigner.default"/>
```

To add a tray for a portal component you must specify the *tray* and *title* attribute:

```
<component name="StyleSheetDesigner.default"
                    title="StyleSheetDesigner.default"
                    trayType="SAPTrayD3"/>
```

To define the component as isolated you must specify the *isoltated* and *height* attribute:

```
<component name="StyleSheetDesigner.default"
                    title="StyleSheetDesigner.default"
                    trayType="SAPTrayD3"
                    isolated="true"
                    height="800px"/>
```

Portal Development Kit (PDK) Wizards for JBuilder 5.

Index

- What is JBuilder and what are Opentools and PDK Wizards?
- Download and Installation of the PDK-Wizards
- PDK-Wizards Guide

What is JBuilder and what are Opentools?

JBuilder is an Integrated Development Environment (IDE) from Borland for Java programming. The enviroment includes a language sensitive editor, project management, debugging features etc.. The functionality of the JBuilder can be extended by third parties. Borland calls these extensions Opentools. For faster and easier development of Portal Components/Services the Portal Development Kit (PDK) provides Opentools for JBuilder Version 5. The Opentool extensions for the PDK are called **PDK Wizards**. The PDK Wizards support the creation of a portal component, definition of the property files, PAR file creation and instant deployment of the portal component.

PDK-Opentools: Download and Installation Instructions

1. In case JBuilder is running close JBuilder completely.
2. **Delete all previous versions of PDK Open Tools!**
   Remove all Par[...].jar, Zar[...].jar and SAPPortals*.jar files from the folders `lib/ext` and `doc`. (Both folders are under the JBuilder-home directory.)
3. Download the file **SAPPortalParWizards.jar** into the folder `lib/ext` under the JBuilder-home directory. It contains the PDK Wizards.
4. Download the file **ParWizardDoc.jar** into the folder `doc` under the JBuilder-home directory. It contains the documentation of the PDK Wizards.
5. Start JBuilder.
6. Go on with the Guide.

How to develop my portal component

Remote Debugging

Why Use Remote Debugging?

If you want to debug your portal component you have to you use remote debugging because:

1. Portal components are not servlets.
2. Tomcat (servlet engine) uses the environment variables different than the built in Tomcat in JBuilder.

How to Debug Remotely

Preparation

Make sure that the `bin` folder of the JDK is listed in the %path% environment variable. You can check your %path% environment variable by typing 'set path' in a command shell. You can change the environment in the following way:

- Select the 'My computer' icon on your desktop with the right mouse button. Then select the menu entry 'properties'. A window appears. Select the tab 'Advanced' in this window. You see the 'Environment Variables' button, which allows you to modify your path (e.g. **append 'c:\jbuilder5\jdk1.3\bin'**).
- Alternatively, you can modify the batch file 'tomcat.bat' by adding the following line after the original 'set path' command: **set path=%path%;c:\jbuilder5\jdk1.3\bin**

Necessary Changes in the Tomcat Startup Batch File:

When you debug, you have to start the Java virtual machine with some extra parameters so that JBuilder can attach itself to a running Java process. You can simplify debugging by modifying Tomcat's main batch file (%TOMCAT_HOME%\bin\tomcat.bat):

- Search for "runServer" and add a line for "debug":

```
if "%1" == "run" goto runServer
if "%1" == "debug" goto debugServer
if "%1" == "ant" goto runAnt
```

- Search again for runServer and add a debugServer option:

```
:runServer
rem Running Tomcat in this window
if "%2" == "-security" goto runSecure
java %TOMCAT_OPTS% -Dhttp.proxyHost=proxy -Dhttp.proxyPort=8080 -
Dtomcat.home="%TOMCAT_HOME%" org.apache.tomcat.startup.Tomcat %2 %3 %4 %5 %6 %7 %8 %9
goto cleanup
```

```
:debugServer
rem Debugging Tomcatin this window
```

```
java -classic %TOMCAT_OPTS%-Xdebug -Xnoagent-Djava.compiler=NONE-
Xrunjdwp:transport=dt_socket,address=5000,suspend=n,server=y
     -Dhttp.proxyHost=proxy -Dhttp.proxyPort=8080 -Dtomcat.home="%TOMCAT_HOME%"
org.apache.tomcat.startup.Tomcat  %2 %3 %4 %5 %6 %7 %8 %9
goto cleanup
```

Note: Special setting for following Tomcat versions apply:

Tomcat Version 3.3:

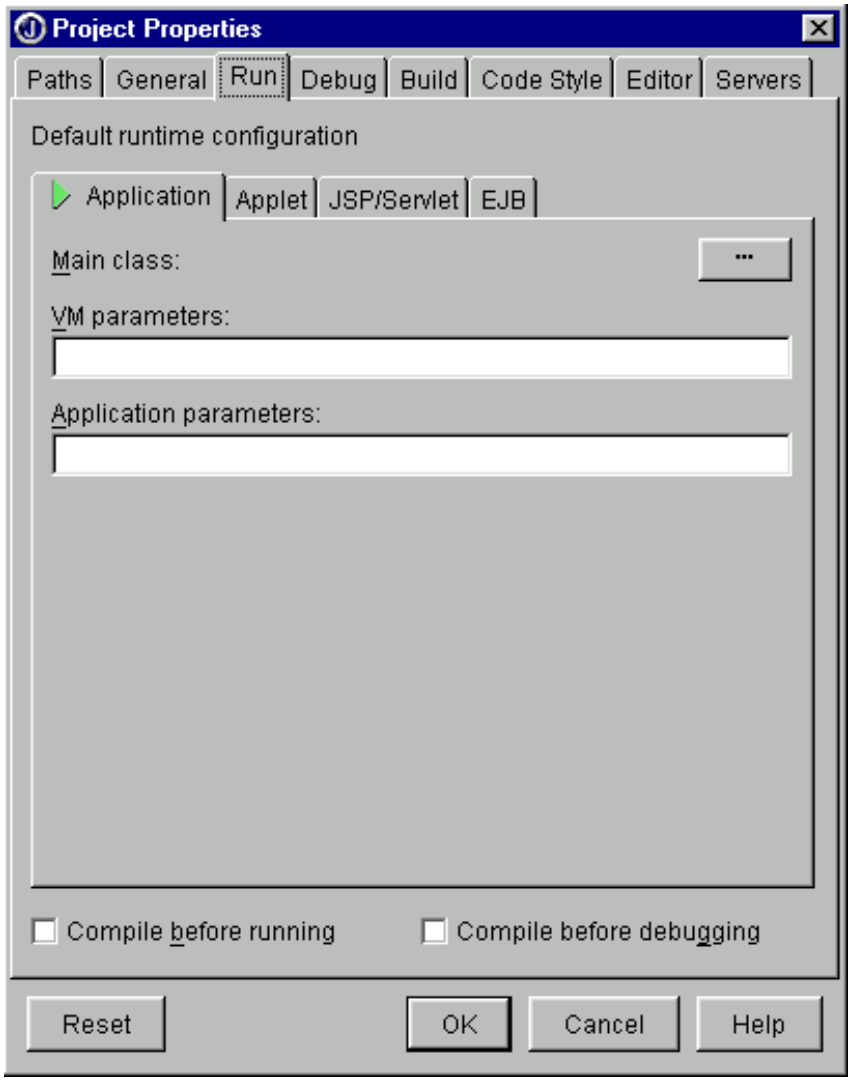
```
The Tomcat development team changed the name of the main class from 'Tomcat' to
'Main' so you  have to use:

:debugServer
rem Debugging Tomcatin this window
java -classic %TOMCAT_OPTS%-Xdebug -Xnoagent-Djava.compiler=NONE-
Xrunjdwp:transport=dt_socket,address=5000,suspend=n,server=y
      -Dhttp.proxyHost=proxy -Dhttp.proxyPort=8080 -Dtomcat.home="%TOMCAT_HOME%"
org.apache.tomcat.startup.Main %2 %3 %4 %5 %6 %7 %8 %9
goto cleanup
```

Tomcat Version 3.3 Beta 2:

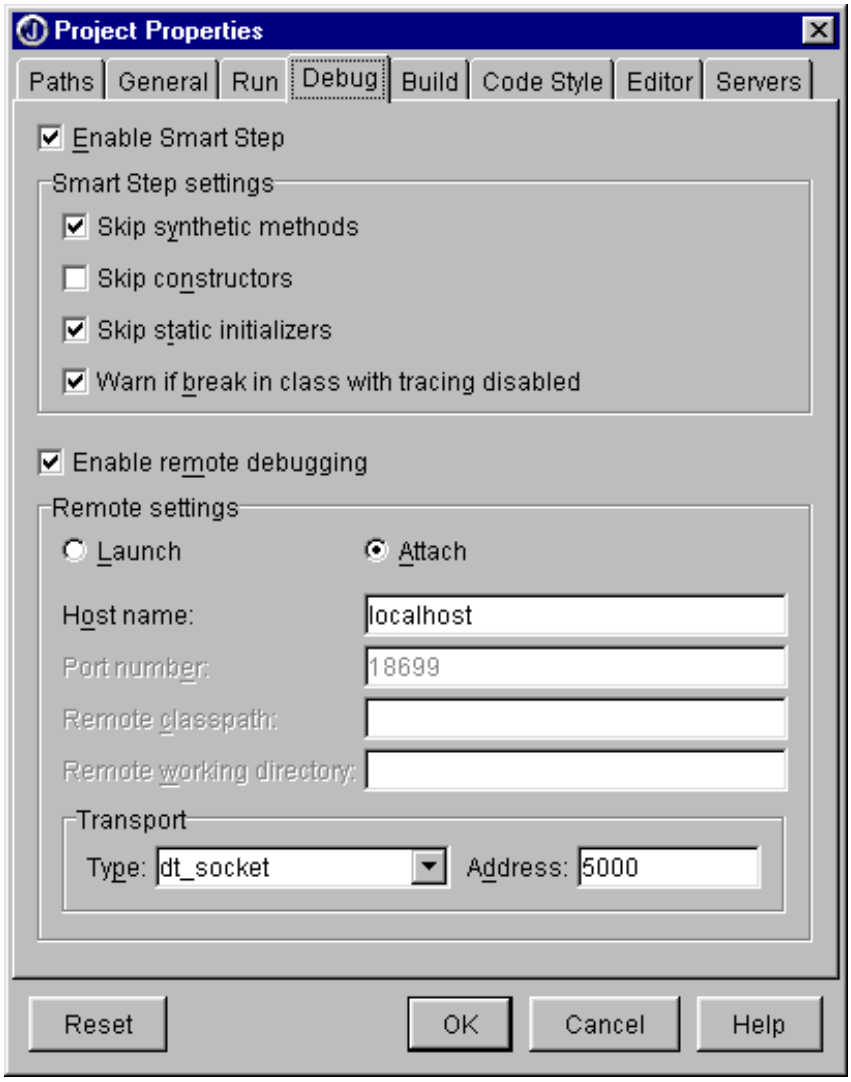
```
The Tomcat development team changed the name of the main class and uses a variable
%_MAIN% for org.apache.tomcat.startup.Main.
If you use this version you have to use:

:debugServer
rem Debugging Tomcat in this window
java -classic %TOMCAT_OPTS%-Xdebug -Xnoagent-Djava.compiler=NONE-
Xrunjdwp:transport=dt_socket,address=5000,suspend=n,server=y
     -Dhttp.proxyHost=proxy -Dhttp.proxyPort=8080 -Dtomcat.home="%TOMCAT_HOME%"
%_MAIN% start %2 %3 %4 %5 %6 %7 %8 %9
goto cleanup
```

Necessary Settings in JBuilder 5:

1. Select following commands from the command menu:
   `Project - Project Properties`
2. Select the `Run` tab and there the `Application` tab.
3. Remove the checkmark from both checkboxes: `Compile before running` AND `Compile before debugging`



4. Select the tab `Debug`
    - Check "Enable remote debugging",
    - Select the radio button "Attach"
    - Select "dt_socket" as Type of Transport (Address should be "5000")



Starting the Debug Session

1. Start the Tomcat Servlet Engine in debug mode.
2. Start the PDK and deploy the component you want to debug.
3. Start the debug session in JBuilder (`Run - Debug Project`) and your break points.
4. Start your component in the PDK.

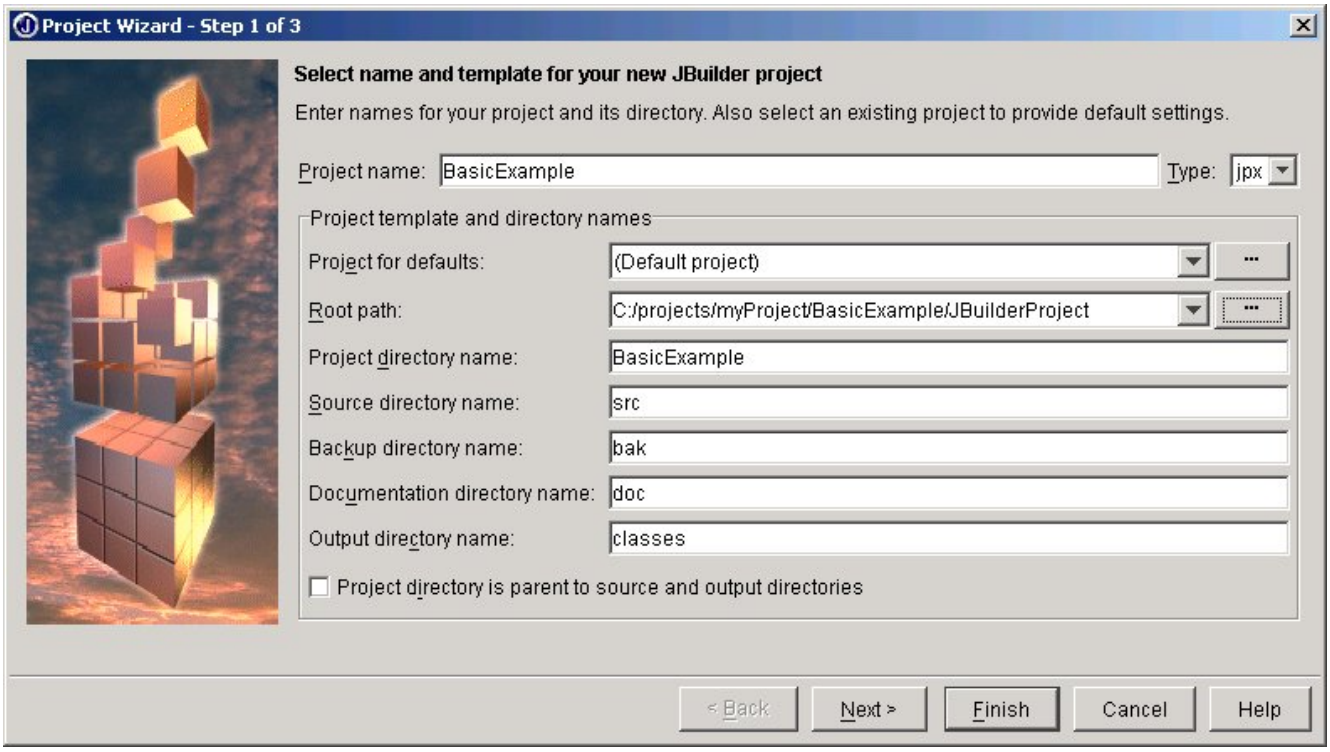
How to develop my portal component

Setting up a JBuilder Project

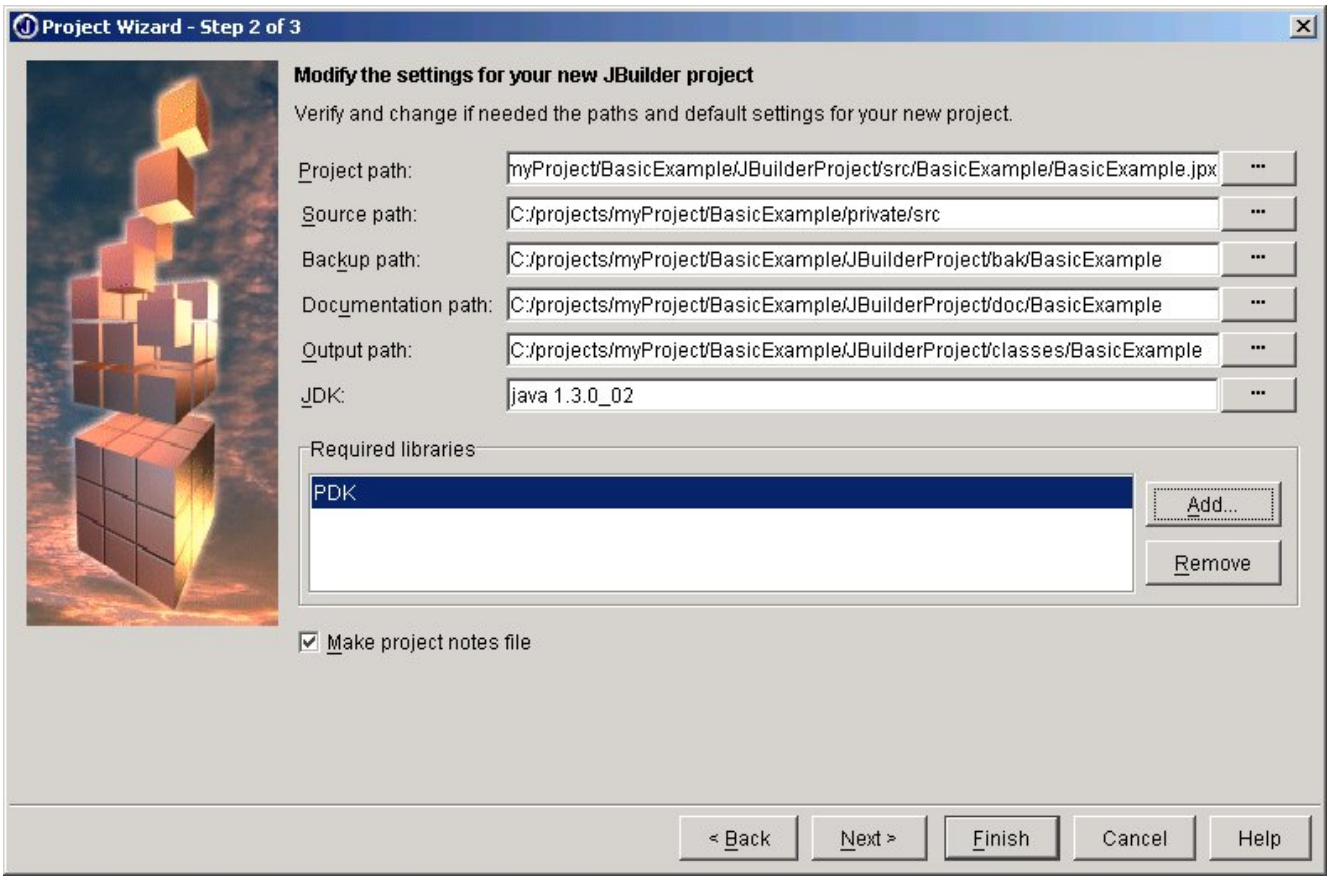
Creating a New Project

As an example we build the BasicExample project.

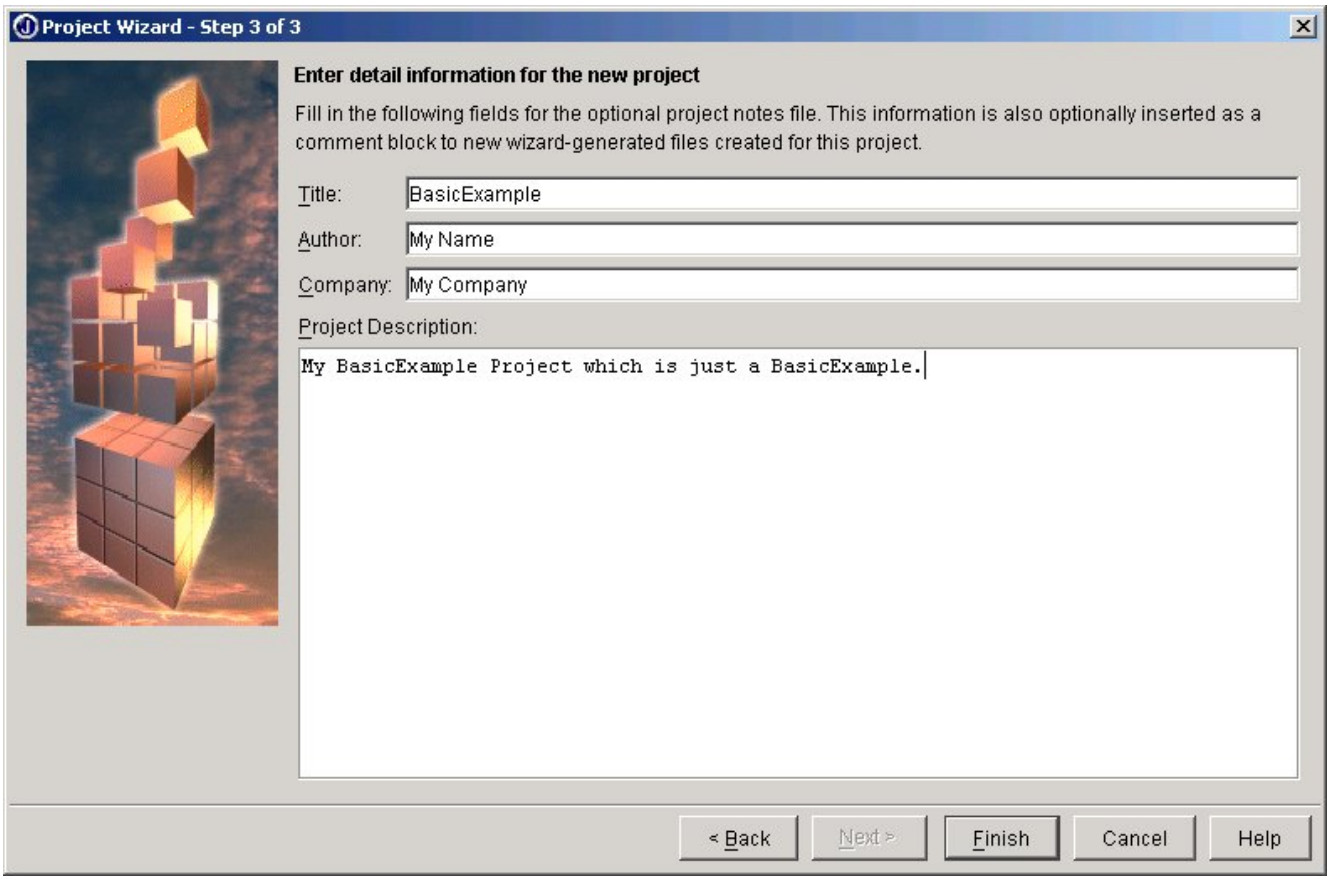
- File -> New Project
  - Project Name: "BasicExample"
  - Root path : "c:/projects/myProject/BasicExample/JBuilderProject/"
  - ***Next***



  - Source path: "C:/projects/myProject/BasicExample/private/src"
  - Required libraries -> Add -> [Select or define new - see: PdkLibraries]
    For the following examples, you will need the jar-Files:
    - htmlb.jar
    - htmlbbridge.jar
    - prtapi.jar
  - ***Next***



- Detailed information: [enter any text]
- ***Finish***



- File -> New -> Class
  - Package: "com.sapportals.pdk.htmlb.basicexample"
  - Class name: "ExampleOne"
  - OK
- For more information -> see the examples

Building and deploying Portal Components with Ant

Index

- Building and deploying with Ant
  - Building with Ant
  - Deploying with Ant

Building and deploying with Ant

Ant is an open source java-based build tool. Through its extensibility Ant can be used to build and deploy portal components. Though we provide IDE plugins for most .par-file related tasks, you can also use Ant to accomplish the task of building and deploying a Portal Archive. (Please note that the build files referenced here where tested with Ant Version 1.3)

Building with Ant

For using Ant as the build tool to create par-files you first have to go through the following installation steps:

1. Download and install Ant.
2. **Add ANT_HOME** to your Environment variables (e.g. `SET ANT_HOME=c:\jakarta-ant`)
3. **Modify your PATH** variable to point to the directory that contains the executable Ant file (e.g. `SET PATH=%PATH%;c:\jakarta-ant\bin`). This is needed to start Ant from any directory.
4. **Create directory** `cfg` inside your component's directory. It has to reside at the same directory level as 'public' or 'private'.
5. **Download** the following three files and put them in the previously created `cfg` directory:
   - build.xml (The main build file for Ant)
   - global.properties (Some additional parameters for `build.xml`)
   - libraries.properties (Contains classpath settings to the portal libraries)
6. **Edit the file global.properties.** This file contains settings of the form `parameterName=parameterValue`. You have to edit the following three parameters:
   - `component`: Here you have to put the name of your component.
   - `par.name`: This will be the filename of the par.
   - `portal_lib_root`: The base directory of your portal (e.g.

`C:\Tomcat`).

After performing step 1-5 you can now easily use Ant to create your par file:

- **Execute Ant** by opening a command prompt, change to the previously created `cfg` directory and simply type `ant` without any parameters. Ant will then search for a file named `build.xml` and will execute the default target which is the target to create the par file. If the build succeeds you can find the par file in your component's directory.
- **Upload** your component into your portal (e.g. with the ComponentManager)

Deploying with Ant

With two more configuration steps you can also automate the work for deploying a par-file to the portal. (**Important:** The component `DevelopmentTools` has to be loaded in your portal. If you read this documentation inside your portal, the component is already loaded.)

1. Download the "Ant upload task" and put it in the following directory: `<ANT_HOME>/lib` (where `<ANT_HOME>` is the installation directory of Ant).
2. **Edit** the file `build.xml` (resides in your component's cfg directory) and uncomment the line at the top beginning with `<taskdef name="upload" ...`.If your portal is not running on `localhost` at port 8088, you have also to search for the line beginning with `<upload serverName="...` at the bottom of the build file. Here you can change the attributes `serverName` and `port` to match your portal configuration.

Now you can use Ant to also deploy your component to the portal. This is done in the following way:

- Open a command prompt, navigate to your components `cfg` directory and type `ant deploy`.

This will first build your component into the component's directory and upload it into the portal. This means you don't need to use the ComponentManager anymore to manually upload your component.

What is the Portal Component

A portal component has an application part (e.g. a Java class and/or Java Server Pages (JSP)) and can use resources (e.g. images, language bundles etc.). A portal component needs a property file that tells the portal runtime which class to use when the portal component is called, which services the portal component needs, in which mode the portal component should run - to name a few. All the files that belong to a portal component are packaged into a par file (PAR = Portal Archive). The par file is than deployed into the PDK local portal or finally into the Enterprise Portal (EP).

The PDK provides two methods to create a portal component.

- Abstract Portal Component
  Components using the Abstract Portal Component class offer a lean method to write HTML commands to the web client. The Abstract Portal Component offers basic event handling, so that for larger, interactive components the programming effort is higher.
  Abstract Portal Component example.

- DynPage
  The Page Processor Component returns a DynPage. It provides a more sophisticated event handling and easier access of JSP (Java Server Pages) files.
  DynPage example.

The PDK documentation and examples focus on the DynPage. The easier event handling and the idea to separate the content development (JSP) from the application development (Java) make the DynPage a better choice for components with interaction and changing content.

Development Tools Support

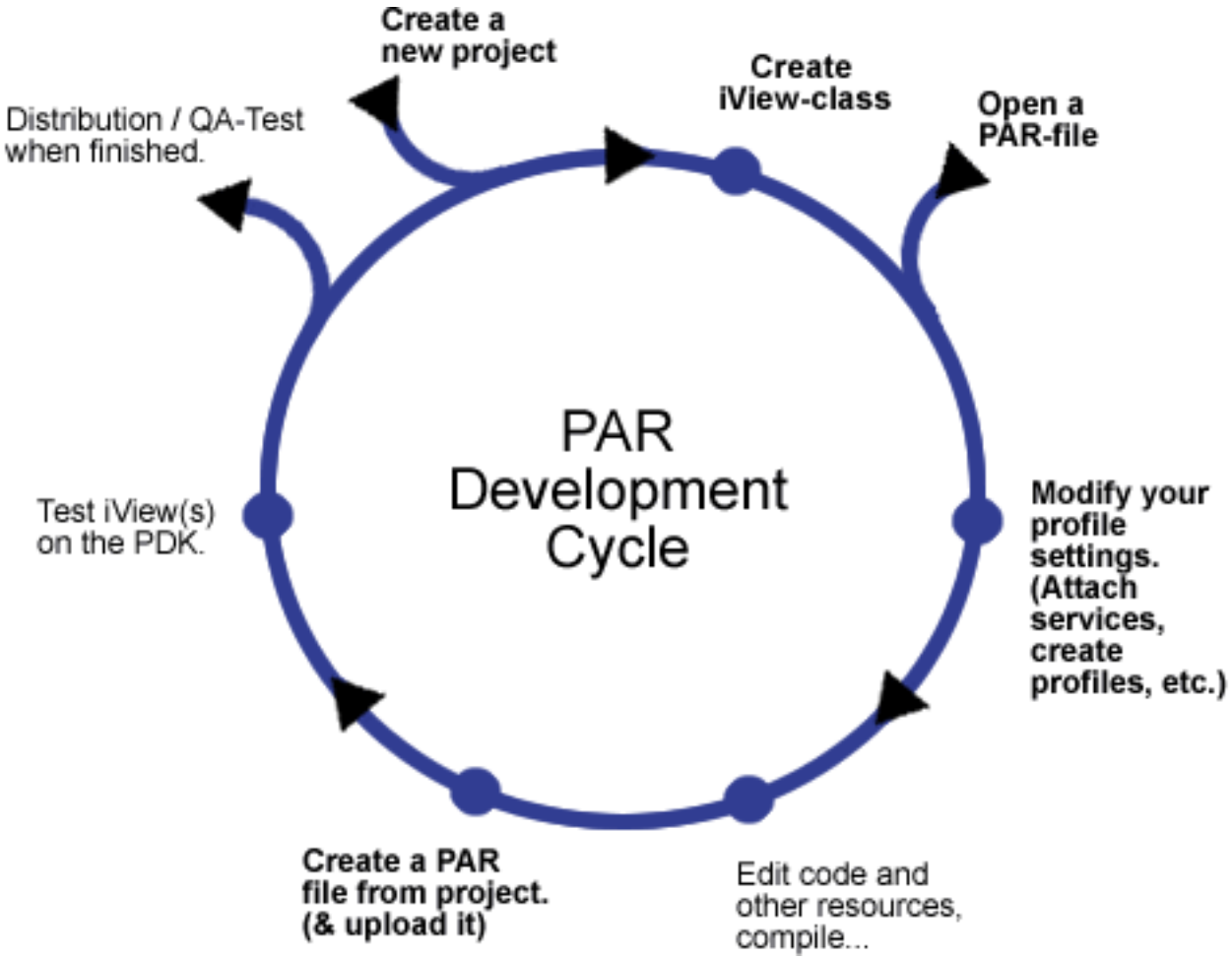
Index

- Portal Archive Development Cycle Using JBuilder
- Short Tutorial: Creating a Hello World PAR
- PDK Wizard Configuration
- New Portal Component Project Wizard
- Open existing PAR Wizard
- New Portal Component Wizard
- Create A Par Wizard
- Profile Wizard
    - Profile Wizard - General Profile Settings tab
    - Profile Wizard - Individual Profile Settings

Portal Archive Development Cycle Using JBuilder

The Portal Development Kit (PDK) wizards for the JBuilder speed up the recurring task of Portal Archive (PAR) file creation. You develop and deploy your portal component without changing the development environment.

The following chart shows a typical Portal Archive (PAR) development cycle. When you click in the text written in bold characters, information about the development step is displayed and how the wizards support you.



Short Tutorial: Creating a Hello World PAR

This tutorial guides you to create a 'Hello World'-PAR file. The estimated time effort is about 5 minutes, assuming that you have the JBuilder PDK wizards already installed and the Portal Development Kit (PDK) on your computer is up and running.

1. Step: Setup of the Wizards. After installation of the PDK wizards start JBuilder. If the PDK wizards have been installed properly you will find additional icons in JBuilder's toolbar and in the menus 'File', 'Project' and 'Tools'. You can skip the setup if you have a default installation of the PDK, e.g. the PDK is reachable by the server address localhost:8080 from your computer. If not, please select **Tools > PDK: Configuration** from the JBuilder menu. Do your settings and press 'OK'. Details about these settings can be found here.

2. Step: Create an portal component project Now we create a new portal component project. To achieve this, select from the JBuilder menu **File > PDK: New Portal Component Project** from the menu. A dialog window opens up. Specify a project name and the folder for the project (or use the default values) and click on the 'OK' button to create the project.

3. Step: Create an AbstractPortalComponent class The 'New Portal Component Project Wizard' creates a new AbstractPortalComponent implementing class for you. The dialog window allows you to specify:

1. ***MyClass*** as Class Name
2. ***mypack*** as Package Name

Leave the 'Set this portal component as default' option checked so that the wizard makes the ClassName entry in the default.properties file. Now press the 'OK' button.

4. Step: Coding JBuilder has generated the minimal implementation of our base class. The doContent(...) method in our MyClass class is empty We use this method to produce the ouptut. With the command line

```
response.write("Hello");
```

we implement the doContent(...) method with a write command that writes the text string 'Hello'.

5. Step: Compilation and Deployment To compile and package our project in a par file we have to click on the symbol  in the tool bar. If the compiler detects any errors the par file creation process is stopped. You have to correct the errors and than start the wizard again. If the compilation was successful the 'Create PAR'-wizard comes up with a dialog window. You can specify a par file name (or use the default) and the the check box 'Deploy portal component to local portal' so that the wizard automatically deploys the par file into the PDK. Press the 'OK' button to start the process.

6. Step: Testing your PAR If you have not changed the default par file name in the 'Create Par File Wizard' dialog window you can call your portal component with the address http://yourHost:yourPort/irj/servlet/prt/portal/prtroot/NewParProject.default

If you can't see 'Hello' in your Browser please check:

- Is your PDK up and running?
- Is your PDK up and running on the same machine as JBuilder you are using right now?
- Have you done the setup (Step 1) right? Try the port number that is entered in your setup dialog by typing the URL http://localhost:*portNumber*/irj . If you can not get the SAP Portal, the port number specified is wrong.
- Is the entry in the default.properties file (folder private/profiles) spelled correctly? The ClassName in the default.properties file match exactly (case sensitive) the name

Go to PDK Wizard Configuration, New Portal Component Project Wizard, Open existing PAR Wizard, New Portal Component Wizard, Create A Par Wizard, Profile

Wizard

How to develop my portal component

PDK Libraries

The libraries needed by the PDK are in sub folders of the %TOMCAT_HOME folder (e.g. C:\tomcat):

```
/lib/apps/
 jCO.jar
 JCOBasicProxy.jar
 SAPProxyUtil.jar

/webapps/irj/WEB-INF/lib/
 ext/servlet-3.1.jar
 prtdispatcher.jar
 prtportalservice.jar
 prtutil.jar

/webapps/irj/WEB-INF/plugins/portal/lib/
 LockServer.jar
 prtapi.jar
 prtcoreservice.jar
 prtjsp_api.jar
 prtlayout.jar
 prtplugin.jar
 servlet.jar

/webapps/irj/WEB-INF/plugins/portal/services/
 adminlogger/lib/adminloggerapi.jar
 cache/lib/cacheapi.jar
 document/lib/documentapi.jar
 epcfloader/lib/epcfloaderapi.jar
 exportal/lib/exportalapi.jar
 htmlb/lib/htmlb.jar
 htmlb/lib/htmlbbridge.jar
 inqmy/lib/inqmyxml.jar
 jco/lib/jcoapi.jar
 jcoclient/lib/jcoclientapi.jar
 keystore/lib/keystoreapi.jar
 laf/lib/lafapi.jar
 landscape/lib/landscapeapi.jar
 navigation/lib/navigationapi.jar
```

```
 pcdadmin/lib/pcdadminapi.jar
 pcdmanager/lib/pcdapi.jar
 pcdmanager/lib/pcdmanagerapi.jar
 pcdmanager/lib/private/pcdcore.jar
 pcdservice/lib/pcdserviceapi.jar
 urlgenerator/lib/urlgeneratorapi.jar
 usermanagement/lib/iaik_jce.jar
 usermanagement/lib/iaik_ssl.jar
 usermanagement/lib/private/um.jar
 usermanagement/lib/tc_sec_api.jar
 usermanagement/lib/tc_sec_core.jar
 usermanagement/lib/tc_sec_jni.jar
 usermanagement/lib/umapi.jar
 usermanagement/lib/usermanagementapi.jar
 xsltransform/lib/xsltransformapi.jar

/webapps/irj/WEB-INF/plugins/portal/system/lib/
 inqmyxml.jar
 prtconnection.jar
 prtcore.jar
 prtgluer.jar
 prtlogger.jar
```

How to Develop a Portal Component

Portal Component Structure

To deploy a portal component into the portal you need a Portal Archive (PAR) file. The par file is a ZIP file with the extension par. A portal component has several files in order to work properly. It has a properties file and a Java class and can have JSP's and resources. The different files have to placed in certain folders in the par file so that the servlet machine can deploy the par file correctly.
The par structure (required directories are highlighted with bold characters):

```
*.par
  •
  ••••meta-inf            Automatically generated.
  •
  ••••public              All files and folders under this folder are deployed as
public. This is usually resources,
  •        •images        images,
  •        •css           css,
  •        •scripts       JavaScripts etc.
  •
  •
  •
  ••••private             The private folder is deployed in an area that cannot be
accessed from “outside“
            •             via Web browser.
            •
            •srclib       Contains the sources of the portal component as jar files
with the component name.src.jar.
            •             Example: BasicExample.src.jar
            •             The sources are not required to run a portal component.
            •
            •profiles     Contains the properties files of the portal component
            •
            •lib          Contains the class files as a jar file with the component
name.jar. Example: BasicExample.jar
            •
            •pagelet      Contains the Java Server Pages (JSP) you use.
            •
            •buildfiles   Contains the Ant building files.
            •             The Ant building files are not required to run a portal
component.
```

You can extend the folder structure in the public and the private area. All folders in the **public** area can be accessed by the web server. The web server has no access to the **private** area.

The sub folders of the **public** folder are copied to:

```
[%TOMCAT_HOME]\webapps\irj\resources\[par file name]
```

The sub folders of the **private** folder are copied to:

```
[%TOMCAT_HOME]\webapps\irj\WEB-INF\plugins\portal\resources\[par file name]
```

How to Develop My Portal Component

Basic Example 1 - *Hello World* - as "DynPage"

This example will show the text *Hello World* . The user interface is implemented with HTML Business for Java extending the class DynPage.
Please keep in mind, that we recommend to use JSP with the HTML Business taglib to define the user interface. This example shows how to use the HTML Business class library directly and therefor does not use JSP.

Click to run the example.

Scenario

This example has only one screen where the text *Hello World* is displayed.

Components for this Scenario

Source Files located at /private/src	Description
ExampleOneDyn.java	Implementation of the portal component.
Property File located at /private/profiles	**Description**
default.properties	Default property file. In this file you must specify the services you are using in the your component.
ExampleOneDyn.properties	Property file to start this example. It contains the class name of your component.

Implemented classes for this examples

First step is to implement a class that extends the class PageProcessorComponent. This class is called from the Portal Runtime for Java (PRT) every time a content request is performed. In this class you have to overwrite the method *getPage()* and you receive a new instance of the DynPage.
In this method you load another class that extends the class DynPage or the class JSPDynPage:

```
public class ExampleOneDyn extends PageProcessorComponent {

  public DynPage getPage() {
    return new MyDynPage();
  }
}
```

The main class responsible for the portal components is the extended DynPage class. In this example the DynPage is implemented as an inner class of `ExampleOneDyn`. `ExampleOneDyn` is the class name we have to define in property file so that the PRT can start the class. In the implementation you have to overwrite the methods:

- **doInitialization** -> called only during initialization of an instance of this component.
- **doProcessAfterInput** -> called when you send data to the server: values, events and so on.
- **doProcessBeforeOutput** -> always called. This method is responsible for the graphical elements of the components. In our example only doProcessBeforeOutput is used.

```
public class DynPageOne extends DynPage {

  public void doProcessBeforeOutput() throws PageException {
    Form myForm = this.getForm(); // get the form from DynPage
```

```
    TextView myText = new TextView(); // set the TextView
    myText.setText("Hello World!");
    myText.setDesign(TextViewDesign.LABEL);

    myForm.addComponent(myText); // add myText to the Form
  }

  public void doInitialization() {
  }

  public void doProcessAfterInput() throws PageException {
  }

}
```

Note: This example does not take full advantage of the DynPage features. We use only one method (`doProcessBeforeOutput`) to create output and still have to overwrite the methods `doInitialization` and `doProcessAfterInput` which we leave empty and we have no event handling. An abstract portal component would need only `doContent` to generate the "Hello World!" output.
In more complex portal components with event handling and sophisticated graphical user interface the advantages of the DynPage become obvious. That is the reason why the PDK documentation and examples are mainly based on the DynPage and JSPDynPage technology.

In the `doProcessBeforeOutput` method we create a form and use a the HTML Business for Java (HTMLB) control `TextView` to create a text output. The `TextView` control has attributes like `text` (= the text that will be displayed) and `design` (=how the text will be displayed). You will learn more about HTMLB in the following chapters and examples. If you already curious you find more details in the document HTMLB Reference.
After we set all the `TextView` attributes we add the `TextView` control to the form. All elements added to the `form` are displayed in the order you added them.

Define the Property Files

The Portal Runtime for Java (PRT) needs some technical information about the portal component. This information is provided in a property file. The property file is created in the folder *private\profiles* of your project. For each component of your project you have to define a separate profile file.
This example comes with the property file named ExampleOneDyn.properties. In this property file the class name of the loader class is specified:

```
ClassName=ExampleOneDyn
```

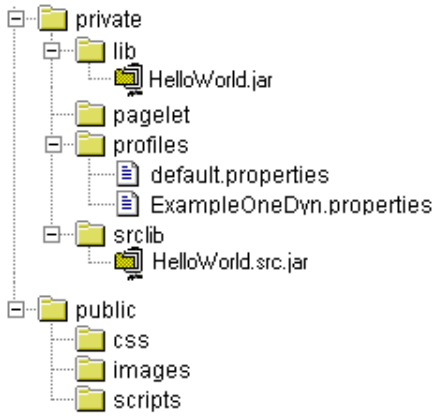
All components of your project are sharing some properties defined in the property file named *default.properties*. In this property file you specify some properties that are valid for all the components. For example you specify here with the parameter *ServicesReference* the services you are using in your components. In this example the service htmlb is used to build the graphical user interface. The declaration in the *default.properties* file looks like this:

```
ServicesReference=htmlb
```

Deployment

In order to run your portal component in the portal you have to deploy your component. First step is to create a .par file (PDK wizards for JBuilder or Ant Building Tool) and than upload the .par file with the `ComponentManager` in the `DevTools` section of the PDK.

Your .par file should have following structure:



Explanation:

private	Description
lib	.jar file with the class files of the component.
pagelet	JSP files. Not used in this example - empty.
profiles	Property files for the component.
srclib	.jar file with the Java source of the components. The source is not necessary for deployment and can be empty.
public	**Description**
css	Style sheets for the component.
images	Default folder for images (GIF or JPEG files). You can store images in another folder.
scripts	JavaScript files

Errors

In case your component throws exceptions when you execute it here are the most common error messages:

Error message:

> Exception caught :
> delegatedplugin
>
> Original exception caught :
> Error in init method
> Component : null
> Component class : null
> User : null
>
> Original exception caught :
> Could not instantiate implementation class MyDyn.ExampleONEDyn of portal component DynParOne.default
> because: Could not find implementation class
>
> Original exception caught :
> com.sapportals.pdk.tree.tree
> java.lang.ClassNotFoundException: MyDyn.ExampleONEDyn

Reason and Solution:

> Mistyped entry in the component property file ExampleOneDyn.properties.
>
> Mistyped entry:
>
> > ClassName=ExampleO**NE**Dyn
>
> Correct entry:

ClassName=ExampleO**ne**Dyn

The entry is case sensitive. The specification of the ClassName in the property file has to be exactly the same as the Java class.

Error message:

Original exception caught :
Error in init method
Component : null
Component class : null
User : null

Original exception caught :
Could not instantiate implementation class MyDyn.ExampleOneDyn of portal component DynParOne.default because: Linkage error while loading implementation class

Original exception caught :
com/sapportals/portal/htmlb/page/PageProcessorComponent
java.lang.NoClassDefFoundError: com/sapportals/portal/htmlb/page/PageProcessorComponent

Reason and Solution:

Missing htmlb service reference in the default.properties file. Entries necessary in default.properties are:

ServicesReference=htmlb
tagLib=/SERVICE/htmlb/taglib/htmlb.tld

Note:
The `tagLib=` entry is essential when you use the tag library of HTMLB in a JSP. In this example we do not use the tag library so it is not necessary but also does no harm.

Error message:

ParseException: ID17195: Error in parsing taglib tag in web.xml or .tld file of the taglib library.

Reason and Solution:

This error will only occur when you use JSP with HTMLB controls. The reason is a mistyped declaration of the tag library in the JSP verses the default.properties file.

Declaration in default.properties

tag**L**ib=/SERVICE/htmlb/taglib/htmlb.tld

Declaration in JSP - mistyped

<%@ tag**l**ib uri= "taglib" prefix="hbj" %>

Correction:

<%@ tag**L**ib uri= "taglib" prefix="hbj" %>
tagLib is written with a capital L.

Portal Development Kit (PDK) Services and Profiles

Introduction

Services support the developers to accomplish portal typical tasks. Offered services are:

- HTML Business for Java (HTMLB).
  Generate browser independent HTML pages in SAP Style. HTMLB provides powerful controls like chart, table view, tree etc. to develop sophisticated and interactive graphical user interfaces.
- Internationalization.
  Access the required language resource files for multilingual applications.
- Portal Data Viewer (PDV).
  Present data from different sources (XML, JDBC query etc.) in tabular form.
- Logger.
  Log events for diagnosis or statistics.
- JCo.
  JCo connects you to SAP systems.
- User Management.
  Retrieve User Mapping information.
- Enterprise Portal Client Framework (EPCF)
  Eventing between iViews on the web client (browser).

The "Portal Services" section in the PDK documentation covers the available services and how you can create your own service.

Profiles define the behavior of the portal component. The profile is a set of properties stored in a property file located in the Portal Content Directory. Each property has a value and a set of attributes. Some properties are predefined by the iView Runtime for Java. Additional properties are defined to store portal component specific information. For more information see Profiles.

How to Develop My Portal Component

Basic Example 2 – *Hello World with Properties* – as "DynPage"

The second DynPage example shows you how to read data from the properties file. The set up of the methods for the DynPage and the property file are similar to the first "Hello World" example.

Click to run the example.

Prerequisites

You can find more information about accessing profiles in the document "Access of Profiles" and about portal component properties in the document Portal Component Properties. Both documents are under the `Basic Portal Services Documentation` - `Accessing a Profile` section in the PDK.

This example also uses HTML Business for Java (HTMLB) to create a graphical user interface. For more information about HTMLB, see the documents HTMLB Index and HTMLB Reference.

Components for this Scenario

Source Files located at /private/src	Description
ExampleTwoDyn.java	Implementation of the portal component.
Property File located at /private/profiles	**Description**
default.properties	Default property file. In this file you must specify the services you are using in the your component.
ExampleTwo.properties	Property file to start this example.

Implemented classes for this examples

As in the first "Hello World" example you implement a class that extends the class PageProcessorComponent. This class is called from the Portal Runtime for Java (PRT) every time a content request is performed. In this class you have to overwrite the method *getPage()* and you receive a new instance of the DynPage.
In this method you load another class that extends the class DynPage or the class JSPDynPage.

We call the call ExampleTwoDyn and do all the processing in the `doProcessBeforeOutput` method. You have to overwrite the methods `doInitialization` and `doProcessAfterInput` as well. We do not need the methods in the example so the methods are empty.

In this example we improve the output visually by adding a group box around the text. A group box has a title bar and a frame. You can set the width, color and title of the group. HTMLB basically supplies controls to create a graphical user interface. For more information about the HTMLB controls see HTMLB Reference.

How to Develop My Portal Component

Basic Example 3 - *Hello World with Properties to be Saved* - as "DynPage"

This example shows how to build a graphical user interface and to handle events. In "Basic Example 1" we sent the text "Hello World" directly to the web client. In "Basic Example 2" we put the text into a group. Now we also add an input field and a button to the user interface.

Click to run the example.

Scenario

This example creates a dialog window where you can enter a name and submit the input. The submit generates an event that is handled by the DynPage. The name is stored in the property file and displayed in the dialog window.

Components for this scenario

Source Files located at /private/src	Description
ExampleThreeDyn.java	Implementation of the portal component.
Property File located at /private/profiles	**Description**
default.properties	Default property file. In this file you must specify the services you are using in the your component.
ExampleThreeDyn.properties	Property file to start this example.

Implemented classes for this examples

As in the first and second "Hello World" example you implement a class that extends the class PageProcessorComponent. This class is called from the Portal Runtime for Java (PRT) every time a content request is performed. In this class you have to overwrite the method *getPage()* and you receive a new instance of the DynPage.
In this method you load another class that extends the class DynPage or the class JSPDynPage.

We call the call ExampleThreeDyn and do all the processing in the `doProcessBeforeOutput` method. You have to overwrite the methods `doInitialization` and `doProcessAfterInput` as well. We do not need the methods in the example so the methods are empty.

In addition to the second "Hello World" example we add to our user interface:

Control	Description
label	A label is a text field that is used in combination with an input field - the label is the description of the input field.
input field	Input field to allow user input.
button	Used as "Submit" button. When the user finishes his input in the input field, he can click on the button to submit his input. The button generates an event when clicked. You define for the `onClick` event the method `sendName`. Now the advantage of the DynPage comes in play. We only have to define a method `onSendName` and the DynPage directs the event directly to this method. See Building a DynPage for more details on event handling.

For more information about the HTMLB controls, see HTMLB Reference.

Defining an Event for a Button

In the `doProcessBeforeOutput`method you set up all control for the user interface. When you define the button you set the attribute `myButton.setOnClick` to `sendName`. This is all you have to do when setting up the control. Now you have to define the method `onSendName` (the `on...` is added by the DynPage and the following letter after on is converted to a capital letter. If you set the attribute `myButton.setOnClick` to `onSendName` the method name is not converted).
In the `onSendName` method you read the content of input field and set the content in the property file.

The onSendName method

To retrieve the user input out of the input field you have to create a new input field and initialize it with the submitted data (`InputField myInputField = (InputField) this.getComponentByName("name");`). The parameter `"name"` in the `getComponentByName` method corresponds to the id of the input field (see method `doProcessBeforeOutput` where the input field is created). Now we can read the data from the input field with the `getValue` method. The value is then stored in the property file in the `doProcessBeforeOutput` method with the command `profile.setProperty("LastUser", this.helloUser);` .

When is the onSendName method called?

The method is called immediately after the doProcessAfterInput and before the doProcessBeforeOutput method. See Building a DynPage for more details on event handling.

HTML-Business for Java - Building a JSP DynPage

Content

- JSP DynPage
  - Basic Example with JSP
  - Event Handling
  - Data exchange between JSP DynPage and JSP
  - JSP DynPage and JSP data exchange using a bean
  - JSP DynPage and JSP data exchange using the session object
  - JSP DynPage and JSP data exchange using the context object
  - JSP DynPage and JSP data exchange using the request object
- Error Messages

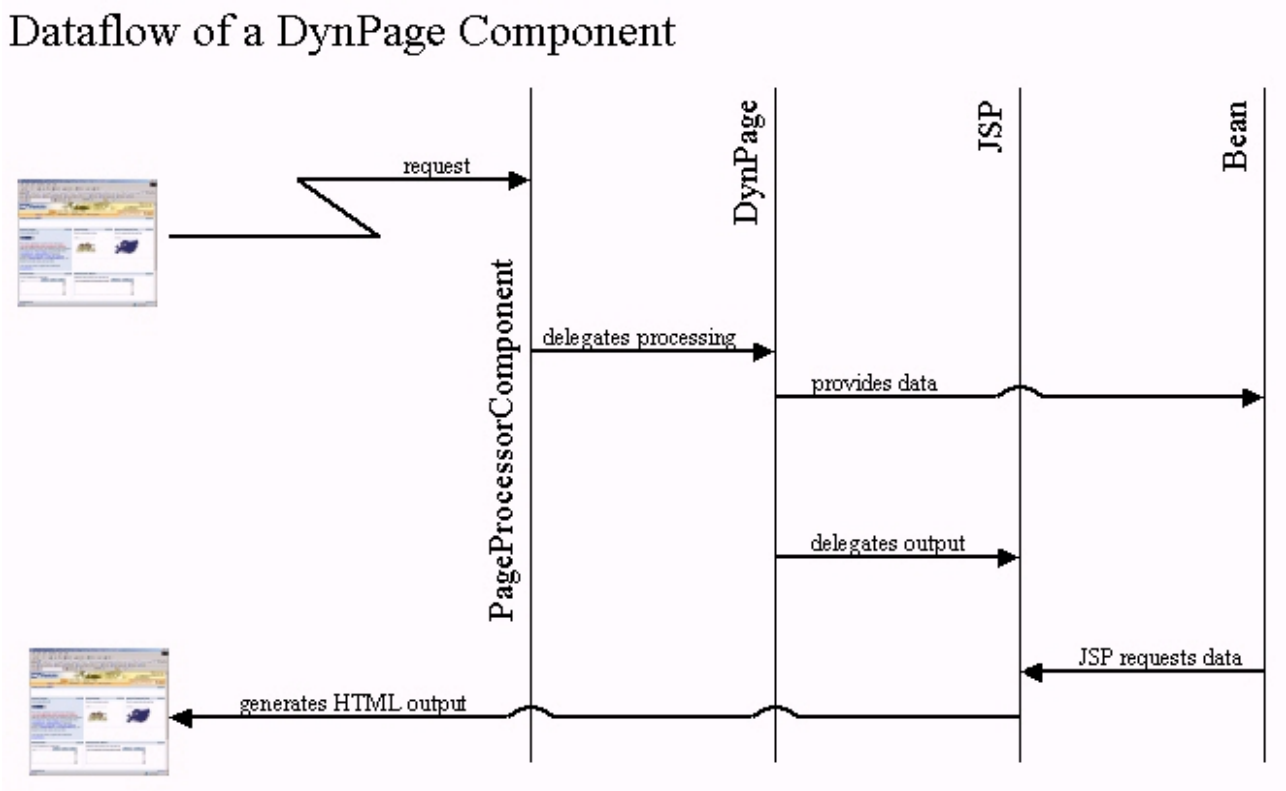
Introduction

To work with this document you need a basic understanding of Java Server Pages (JSP). Sun provides JSP documentation. The Portal Runtime (PRT) has made modifications to the JSP standard. For further details see Java Server Pages (JSP) Support in the Portal Runtime (PRT).

JSP DynPage

The HTML-Business for Java controls are necessary to create a proper GUI. In addition to that a web application needs appropriate event handling to fill the application with life. As described in the controls section there are a number of controls that generate events. If we stay on the JSP level the event handling is pretty basic. The programmer has to take care of the event handling himself (analyzing the received form, getting the sender of the event etc.). In addition the programmer has to take care of the session identifier, an unique identifier that makes sure that the datasets are user specific.

For enhanced event handling and easy session management we introduce the new term **JSP DynPage**. The JSP DynPage uses the Business HTML for Java as API to design the GUI and handles the data and event transfer.



JSP DynPage Basic Example

First step to create a portal component is to define a class that works as loader class - it inherits from the PageProcessorComponent. The created loader class (in the following example named **ExampleOneDyn**) executes the method **getPage()** and returns a unique value of the JSP DynPage we can use (in the following example named **DynPageOne**).

```
package com.mycompany.basicexample;

import com.sapportals.htmlb.page.DynPage;
import com.sapportals.portal.htmlb.page.PageProcessorComponent;


public class ExampleOneDyn extends PageProcessorComponent {

  public DynPage getPage() { // Has to be overridden
    return new DynPageOne(); // Calls the DynPage and returns its value as DynPageOne
  }
}
```

The class **DynPageOne** is extended (in the Java sense) from the DynPage class. Following methods have to be overwritten (in the Java sense):

- **doInitialization**
  Called when the application is started. The call is made when the page is directly called per URI without parameters and no event occurred.
  Usually this method is used to initialize data and to set up models. Be aware of the fact that the `doInitialization` event is also caused when another Portal Component on the same page sends an event.

  Example:
  With the "Personalize" Dialog you can compose a page by grouping several Portal Components together. We have created a page called `myPage` with two Portal Components - A and B. When calling the page `myPage` the `doInitialization` is called from Portal Component A and B followed by the call of the method `doProcessBeforeOutput`. When an event occurs in the Portal Component B (e.g. by clicking on a button), the `doInitialization` method in Portal Component A is called again, while in Portal Component B the method `doProcessAfterInput` followed by the event handling method assigned for the button and finally the `doProcessBeforeOutput` method.

  To create solid Portal Components you must be aware of the fact and check in the `doInitialization` method if the data has already been initialized or the models have been created. Otherwise you Portal Component is always reset to the initial state if an event in another Portal Component occurs.

  The Enterprise Portal treats every Portal Component isolated. In this case an event in one Portal Component does not cause the `doInitialization` event in the other Portal Component on the same page. The PDK can emulate this behavior by setting the ISOLATED flag in the page description XML file to true. The page description XML file in the content folder of the page and defines the

position of the component, height and tray type.

Example for an entry in the XML file:
`<component name="AAA.default" title="AAA.default" height="400" trayType="SAPTrayD3" Position="1"/>`

Example for an entry in the XML file with isolation flag set so that PDK behaves like the Enterprise Portal:
`<component name="AAA.default" title="AAA.default" isolated="true" height="400" trayType="SAPTrayD3" Position="1"/>`

If you use the Page Editor in the DevTools section of the PDK you can set the isolated flag interactively.

- **doProcessAfterInput**
  Called when the web client sends the form to the web server. Except on doInitialization (see above) the call is performed every time an event occurs.

- **doProcessBeforeOutput**
  Called before the form is sent to the web client. The call is performed every time even on doInitialization.

The three methods can be empty shells if the method should not be processed. In our basic example we only use doProcessBeforeOutput to place a textView control.

```
package com.mycompany.basicexample;
import com.sapportals.htmlb.*;
import com.sapportals.htmlb.enum.*;
import com.sapportals.htmlb.page.PageException;
import com.sapportals.portal.htmlb.page.JSPDynPage;


public class DynPageOne extends JSPDynPage {

/*   Constructor  */


        public DynPageOne() {
          this.setTitle("DynPageOne");
        }


/*   Used for user initialization. Called when the application is started   * /

  public void doInitialization() {
        }


/ * Used for handling the input. Generally called each time when an event
    occurs on the client side  */

  public void doProcessAfterInput() throws PageException {
        }

/*   Used for handling the output. This method is always called.
     In our example the JSP makes a textView that displays
     "May the force be with you unknown user".       */

  public void doProcessBeforeOutput() throws PageException {
      this.setJspName("OutputText.jsp"); // set the JSP which builds the GUI
        }
}
```

JSP - OutputText.jsp - that is called by doProcessBeforeOutput

```
<%-- OutputText.jsp --%>
<%@ taglib uri= "tagLib" prefix="hbj" %>
<hbj:content id="myContext" >
  <hbj:page title="An Easy Start">
     <hbj:form>
        <hbj:textView
         id="welcome_message"
         text="May the force be with you unknown user"
         design="HEADER1"
        />
     </hbj:form>
  </hbj:page>
</hbj:content>
```

On first sight it looks more complicated to build a web component with JSP DynPage. The advantages become very obvious when your application grows. Complex web components are easier to read (therefore easier to support and change) and as you will see now how easy it is to add event handling to the JSP DynPage.

JSP DynPage Event Handling

Some HTML-Business for Java controls have an event attribute (e.g. see button). The button for example has an 'onClick' attribute that specifies the name of the method which should handle the event. The event will occur when the appropriate user action takes place - in this case clicks on the button. The name of the method specified with the 'onClick' attribute has to be declared in the JSP DynPage.

HTML-Business for Java: Statement to define the method name:

```
Button1.setOnClick("myClick");
```

JSP DynPage: Declaration of the method that processes the event:

```
public void myClick (Event event) { ..coding.. }
    or
public void myClick (Event event) { ..coding.. }
```

Both declarations are valid. The decision to use the `on...` method declaration could be helpful to make it obvious that this method handles an event.

Note:

With the `on..` declaration method the first letter of the declared event name must be a capital letter.

Example

```
event declaration:   Button2.setOnClick("myClick");
method declaration:  public void onMyClick (Event event) { ..coding.. }
```

If both methods are declared (myClick and onMyClick), only the method **without** "on" will be called. onMyClick will be ignored.

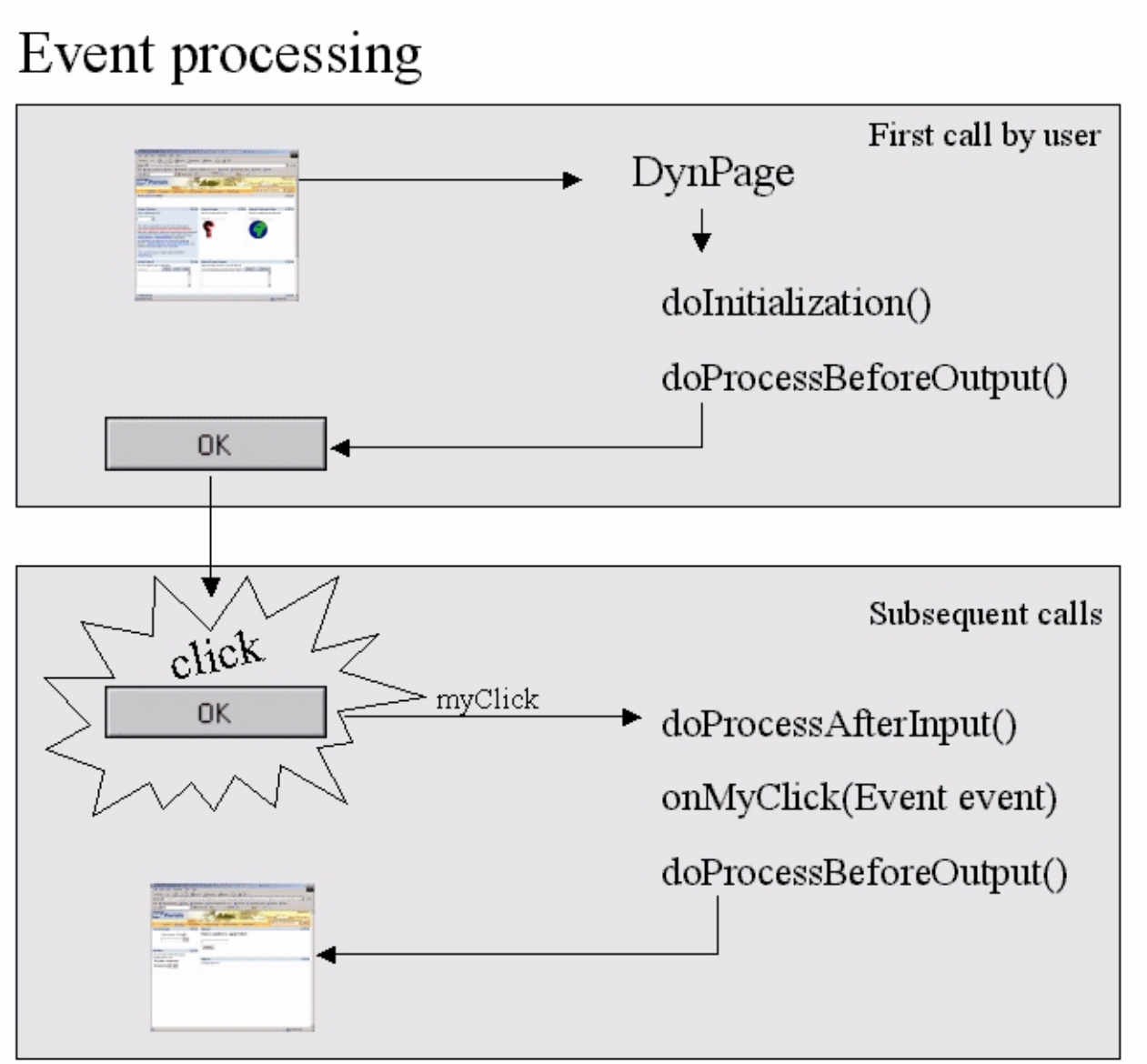
Hint:
Some HTML-Business for Java controls have a '`setOnClient` ' attribute. With this attribute you specify a JavaScript fragment that handles the event on the Web client side. The event is NOT transmitted to the Web server.

If an event occurs it is handled as follows (doInitialization is performed when the application starts).

- **doProcessAfterInput**
  Called when the web client sends the form to the web server.
- **onMyClick**

The event handling method we declared.

- **doProcessBeforeOutput**
  Called before the form is sent to the web client.



Data exchange between JSP DynPage and JSP

The storing methods we discuss in this section are volatile in the sense that the data is lost when the session is over (or even before that). Generally you have to decide if the data you provide in the JSP DynPage should be shared among other users and how long the data must "live". For storing data permanently you can refer to the Profile documentation.
Storing data can be performed using:

- beans
  A bean is defined with set and get methods. The bean can be accessed from the JSP DynPage and the JSP as well as from other users (depending on the scope, see "Usage of Beans").
- session
  Data stored in the http session will be kept by the server as long as the user session is alive.
- context
  The lifetime of data stored in the context cannot be guaranteed. The context can be released any time when the web server needs resources.
- request
  Data stored in the request are kept for the request.

The storing methods in detail:

JSP DynPage and JSP data exchange using a bean

A bean is used to get and set "dynamic" data. The JSP DynPage usually provides the bean with data and the JSP reads the data. The functionality of the basic example is extended by an input field that allows user input. The user input is stored in a bean and than displayed as text by a JSP program.

Following steps are necessary

- create a bean
- initialize the bean
- introduce the bean to the JSP program OutputText.jsp

Declaring a bean in a JSP

The tag **usebean** declares a bean in a JSP**.**

class
Class name of the bean.

id
Identification name of the bean. The id is used to access the bean in scriptlets.

scope
Defines the availability of the bean. Details see "How to use Beans".

attribute	req.	values	default	case sens.	JSP taglib
class	yes	String	none	yes	class="com.sapportals.htmlb.beandemo.myBean"
id	yes	String	none	yes	id="idOfMyBean"
scope	yes	application session request page	none	yes	scope="application"

Bean

```
package bean;


/*
 * A very simple bean whose only purpose is to store a simple String.
 * It as a get and set method to store and recall the string.

 */
public class DynPageNameBean
{
      public String name;


      public String getName ()
      {
         return name;
      }


      public void setName (String name)
      {
         this.name = name;
      }
}
```

The GUI is extended by an inputField and a button to allow the user to enter a string (e.g. user name) and submit the form by clicking the button or pressing Return/Enter on the keyboard.
Following changes to OutputText.jsp are necessary:

- Adding a **form** to allow the definition of a default button.
- Introduce the bean to the JSP program (<jsp:usebean .../>).
- Changing the textView so that it displays the string retrieved from the bean.

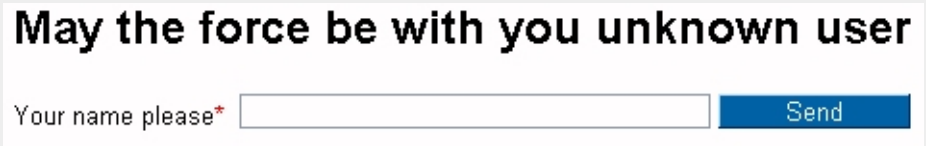
- Adding a **label** telling the user to enter the user name. The label should start in a new line and one line separated from the textView. That is why you find a

 before the label
- Adding an **inputField** "user_name_input" - the JSP DynPage retrieves the data in the input field using **getComponentByName**.
- Adding a **button** which we also the default button. This enables the user to send his input by either clicking on the button or by simply pressing Enter/Return when he finished the input.

```
<%-- OutputText.jsp --%>
<%@ taglib uri= "tagLib" prefix="hbj" %>
<hbj:content id="myContext" >
  <hbj:page title="An Easy Start">
   <hbj:form id="myFormId"
       >
<%-- Declaration of the bean. --%>
<jsp:useBean id="UserNameBean" scope="application" class="bean.DynPageNameBean" />

<hbj:textView
         id="welcome_message"
         design="HEADER1" >

         <%   welcome_message.setText("May the force be with you "
                                      + UserNameBean.getName());   %>
       </hbj:textView>

       <br><br>

       <hbj:label
         id="label_input"
         text="Your name please"
         design="LABEL"
         required="TRUE"
         labelFor="user_name_input"
        />

<%-- inputfield to allow userinput - the inputfield has the id "user_name_input" which is --%>
<%-- used in the JSP DynPage to access the input field and retrieve the input of the user --%>
       <hbj:inputField
         id="user_name_input"
         type="STRING"
         design="STANDARD"
         width="250"
         maxlength="30"
        />

       <hbj:button
         id="Send_Button"
         text="Send"
         tooltip="Sends my name"
         onClick="onSendButtonClicked"
         width="100"
         design="EMPHASIZED" >

         <% myFormId.setDefaultButton(Send_Button);  %>
        </hbj:button>

   </hbj:form>
  </hbj:page>
</hbj:content>
```

Result of the JSP

Hint: The default string "unknown user" will be set in our JSP DynPage in the following section.



The JSP DynPage has to be changed as well.

- Introduce the bean to the JSP DynPage.
- in **doInitialization** we set a default user name "unknown user".
- Creating a event method **onSendButtonClicked** for the button click in which the status (variable **state)** is set to WELCOME_STATE so that the **doProcessBeforeOutput** selects another JSP file.
- in **doProcessAfterInput** (method is called whenever an event occurs) we request the inputField "user_name_input" from the JSP by using **getComponentByName** to have access to the user input in the JSP DynPage. If the inputField "user_name_input" is not empty the string is stored in the bean.

```
package com.mycompany.basicexample;
/**  introduce the bean */

import bean.DynPageNameBean;


import com.sapportals.htmlb.*;
import com.sapportals.htmlb.enum.*;
import com.sapportals.htmlb.event.Event;
import com.sapportals.htmlb.page.DynPage;
import com.sapportals.htmlb.page.PageException;
import com.sapportals.portal.htmlb.page.JSPDynPage;
import com.sapportals.portal.htmlb.page.PageProcessorComponent;
import com.sapportals.portal.prt.component.IPortalComponentContext;
import com.sapportals.portal.prt.component.IPortalComponentProfile;
import com.sapportals.portal.prt.component.IPortalComponentRequest;

public class DynPageOne extends JSPDynPage
{
    private final static int INITIAL_STATE = 0;
    private final static int WELCOME_STATE = 1;
    private int state = INITIAL_STATE;
    private String name;

        /**
         * Constructor
         */
        public DynPageOne() {
     this.setTitle("Become a Jedi");
        }


        /**
   * Used for user initialization. called when the application is started
         */
  public void doInitialization()
  {
  // create the bean and set a default text value "unknown user

   IPortalComponentRequest request  = (IPortalComponentRequest) this.getRequest();
   IPortalComponentContext myContext = request.getComponentContext();
   IPortalComponentProfile myProfile = myContext.getProfile();

   // new bean object
   UserNameContainer = new DynPageNameBean();
  // set default name
   UserNameContainer.setName("unknown user");
  // store bean in profile for the JSP
    myProfile.putValue("UserNameBean", UserNameContainer);

  // Now we set the state so that we can decide what action to do next
   state = INITIAL_STATE;
        }

        /**
    * Used for handling the input. Generally called on each event
    * we use this method to get the user name and store it in the bean
    */
  public void doProcessAfterInput() throws PageException
  {
  // get the input field from the JSP
   InputField myInputField = (InputField) getComponentByName("user_name_input");
   if (myInputField != null) {
       this.name = myInputField.getValueAsDataType().toString();
   }
   IPortalComponentRequest request
               = (IPortalComponentRequest) this.getRequest();
   IPortalComponentContext myContext = request.getComponentContext();
   IPortalComponentProfile myProfile = myContext.getProfile();
   DynPageNameBean myNameContainer
               = (DynPageNameBean) myProfile.getValue("MyNameBean");
   myNameContainer.setName(name);
  }

        /**
         * Used for handling the output. This method is always called.
   * In our example before the JSP made a textView with "May the force be with you
   * unknown user".
   * We now extend this method that according to the state it either - that is when
   * state = INITIAL_STATE - askes for the user name and calls the user
   * "unknown user" or after init - that is when state = WELCOME_STATE -
   * displays a success message with the username.
         */
  public void doProcessBeforeOutput() throws PageException
  {
         switch (state) {
            case WELCOME_STATE:
                    this.setJspName("OutputSuccessText.jsp");
                    } break;
            default:
                    this.setJspName("OutputText.jsp");
                    break;
        }
```

```
  /**
   * this method handles the event of the button. The event is fired either when
   * the user clicks on the button or presses Return/Enter when he is in the
   * inputField (since we defined the button as default button).
   * In this method we set the state to WELCOME_STATE so that on the following
   * doProcessBeforeOutput (which is called immediatly after this method)
   * a success message is displayed
   */
  public void onSendButtonClicked (Event event) throws PageException
  {
        state = WELCOME_STATE;
  }
}
```

The only thing missing now is OutputSuccessText.jsp which should send the personalized message to the user. The usage of the bean has been already shown in OutputText.jsp so OutputSuccessText.jsp is very small and creates a textview with the username in it.

```
<%-- OutputSuccessText.jsp --%>
<%@ taglib uri= "tagLib" prefix="hbj" %>
<hbj:content id="myContext" >
  <hbj:page title="Successful processing">
   <jsp:useBean id="UserNameBean" scope="application" class="bean.DynPageNameBean" />

      <hbj:textView
        id="success_message"
        design="HEADER1" >

        <% success_message.setText("The force is with you"
                                   + UserNameBean.getName());
        %>
      </hbj:textView>

  </hbj:page>
</hbj:content>
```

Result of the JSP - (assuming that the user responded with "SAPPortals")

The force is with you SAPPortals

Try the example - now that the force is with us it should work. (Note: Portal has to be installed and up and running)

JSP DynPage and JSP data exchange using the session object

Data stored in the http session will be kept by the server as long as the user session is alive.

JSP DynPage

Getting the request object:

```
IPortalComponentRequest request = (IPortalComponentRequest)
this.getRequest();
```

To store a value in the session we have to use the command line:

```
request.getComponentSession().putValue("myText", "A short note
in the session");
```

To get the stored value we have to use the command line:

```
request.getComponentSession().getValue("myText");
```

JSP

To store a value in the session we have to use the command line:

```
<%
componentRequest.getComponentSession().putValue("myText","That
is from the JSP"); %>
```

To get the stored value we have to use the command line:

```
<%
componentRequest.getComponentSession().getValue("myText").toString();
%>
```

JSP DynPage and JSP data exchange using the context object

The lifetime of data stored in the context cannot be guaranteed. The context can be released any time when the web server needs resources. We strongly recommend to refresh the stored data (e.g. in the JSP DynPage method doProcessAfterInput, which is called every time an event occurs on the web client) to avoid an exception. The exception will occur when you try to access a value that is already gone (or has never been set).

JSP DynPage

Getting the request object:

```
IPortalComponentRequest request = (IPortalComponentRequest)
this.getRequest();
```

To store a value in the session we have to use the command line:

```
request.getComponentContext().putValue("myText", "A short note
in the context");
```

To get the stored value we have to use the command line:

```
request.getComponentContext().getValue("myText");
```

JSP

To store a value in the session we have to use the command line:

```
<%
componentRequest.getComponentContext().putValue("myText","That
is from the JSP"); %>
```

To get the stored value we have to use the command line:

```
<%
```

```
componentRequest.getComponentContext().getValue("myText").toString();
%>
```

JSP DynPage and JSP data exchange using the request object

The lifetime of data stored in the request is limited to the request only. You have to refresh the stored data (e.g. in the JSP DynPage method doProcessAfterInput, which is called every time an event occurs on the web client) to avoid an exception. The exception will occur when you try to access a value that is already gone (or has never been set).

JSP DynPage

Getting the request object:

```
IPortalComponentRequest request = (IPortalComponentRequest)
this.getRequest();
```

To store a value in the session we have to use the command line:

```
request.getNode().putValue("myText", "A short note in the
request");
```

To get the stored value we have to use the command line:

```
request.getNode().getValue("myText");
```

JSP

To store a value in the session we have to use the command line:

```
<% componentRequest.getNode().putValue("myText","That is
from the JSP"); %>
```

To get the stored value we have to use the command line:

```
<% componentRequest.getNode().getValue("myText").toString();
%>
```

Error Messages

For more information, see FAQ.

Back to overview

Tutorial Part I

The tutorial covers following topics:

- Creating a portal component with a graphical user interface and event handling.
- Accessing the personalization data (properties).
- Accessing resources.
- Using a portal service.

Click to run the example.

Prerequisites

This example uses the servlet class JSPDynPage from service HTML Business for Java. You must be familiar with this class and how the data is exchanged between the JSPDynPage and the JSP using a bean. This topic is covered by the document "HTML-Business for Java - Building a DynPage".

Scenario

We develop a lottery game on the portal. The user can select a number. This number is saved. A service "draws" a number and compares it with the selected number and notifies the winner. The tutorial starts with the development of the portal component using JSP's. The DynPage shows event handling access of the properties file and how to access a portal service.

Components for this scenario

Source File	Description
Portaloto.java	Methods and Event handling DynPage.
Bean	**Description**
LuckyNumbersBean.java	Contains the selected number.
StringBean.java	Contains text strings that are displayed in the JSP.
JSP	**Description**
WelcomeMessage.jsp	Displays the introduction and a list box to select the number to play.
OutputNumberPicked.jsp	An animated confirmation of the selected number.
OutputText.jsp	Is used to display information or confirmation messages.
Property File	**Description**
Portaloto.properties	Portaloto property file. You must specify the services you are using in the components. It contains also information about the game.

Setting up a JBuilder Project

If you have already developed a portal component you are familiar with the development cycle. If not have a look at the following documentation:

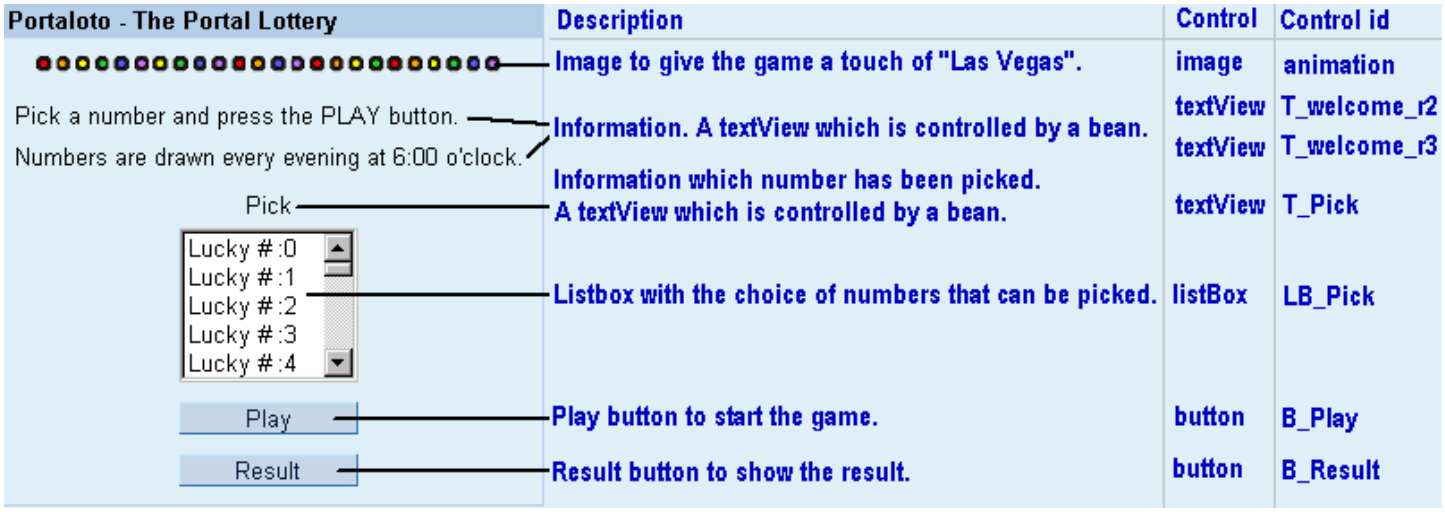
Installation of the plugins	Set up JBuilder for a collaboration with the Portal Development Kit.
Guide	Developing with the JBuilder wizards.
Remote Debugging	Necessary steps to use the JBuilder debugging features to debug a portal component.

Creating the Portal Component

The Portaloto component will display a page which allows the user to pick a number and start the game. The user interface is created with HTML-Business for Java controls in a JSP. The JSP sends the events which are handled by the DynPage. The JSP and the DynPage communicate with beans (see "usage of beans" for details). So lets go.

1. Create the User Interface using JSP's and Define the Events that have to be Handled in the DynPage.

The user interface should look like this:



We use a **group** control with the headline "Portaloto - The Portal Lottery" and place the other controls in the group body. For a better placement of the controls we use the **gridlayout** control. In our example the **gridlayout** has 1 column and 10 rows. The **gridlayout** allows us to center the buttons and the list box and display single text lines.

We create WelcomeMessage.jsp. At the beginning of the JSP we have to do the following steps

- Import the resource package that we can access the resource to get the image
- Declaring the tagLib for HTMLB controls
- Declaring the two beans which we need in the JSP (to get details about the scope of beans click here).

That looks like this:

```
<%-- WelcomeMessage.jsp --%>
<%-- import the resource package - will be used to get the image name --%>
<%@ page import="com.sapportals.portal.prt.resource.IResource" %>
<%@ taglib uri="tagLib" prefix="hbj" %>

<%-- Declare Beans: --%>
<%-- bean.LuckyNumbersBean contains DefaultListModel that supplies the --%>
<%-- listBox LB_Pick with data bean.StringBean contains two text strings --%>
<%-- that are displayed in textView T_Pick --%>
<jsp:useBean id="LuckyNumbers" scope="application" class="bean.LuckyNumbersBean" />
<jsp:useBean id="TextBean" scope="application" class="bean.StringBean" />
```

Then we open a content, a page and a form tag and start coding the HTMLB controls beginning with a **group** and **gridLayout**. The picture above shows the control and the control id we use in our JSP. In the JSP we code the controls "top - down" so with the picture it is very easy to follow the documented code in the WelcomeMessage.jsp file.

So that something will happen when the user clicks on the buttons and the list box we have to define the events and the methods which should handle the events. Following controls have events:

Control	Control id	Event	Event Handling Method	Description
listBox	LB_Pick	onSelect	onSelect	Event will be fired when the user clicks in the list box and selects a number.
button	B_Play	onClick	onPlayClick	Event will be fired when the user clicks on the PLAY button.
button	B_Result	onClick	onResultClick	Event will be fired when the user clicks on the RESULT button.

The event handling methods will be defined in the DynPage which we will create after we are finished with the JSP's.

Next we create a JSP which serves as message dialog page. It displays an information text and has an OK button that the user has to click in order to confirm the message (similar to modal windows in Windows applications). It looks like that:

We use a **group** control with the headline "Portaloto - Infotext" and place the other controls in the group body. For a better placement of the controls we use the **gridlayout** control. The **gridlayout** allows us to center the buttons and text lines.

We create OutputText.jsp. At the beginning of the JSP we have to do the following steps

- Import the resource package that we can access the resource to get the image
- Declaring the tagLib for HTMLB controls
- Declaring the bean which we need in the JSP (to get details about the scope of beans click here).

That looks like this:

```
<%-- import the resource package - will be used to get the image name --%>
<%@ page import="com.sapportals.portal.prt.resource.IResource" %>
<%@ taglib uri= "tagLib" prefix="hbj" %>
<%-- Declare Beans:
<%-- bean.StringBean contains two text strings that are displayed in textView T_Pick --%>
<jsp:useBean id="TextBean" scope="application" class="bean.StringBean" />
```

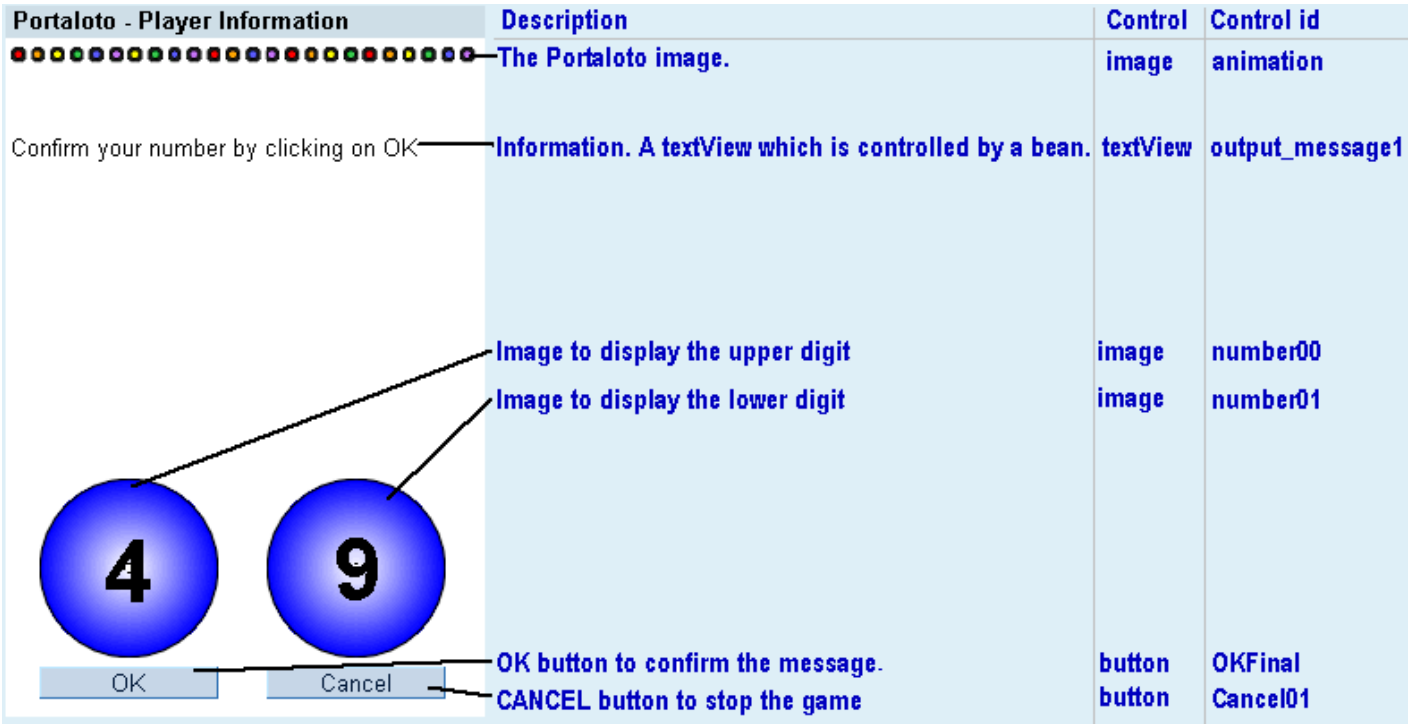
Then we open a content, a page and a form tag and start coding the HTMLB controls beginning with a **group** and **gridLayout**. The picture above shows the control and the control id we use in our JSP. In the JSP we code the controls "top - down" so with the picture it is very easy to follow the documented code in the OutputText.jsp.

For the OK button we define an event:

Control	Control id	Event	Event Handling Method	Description
button	OK01	onClick	onOK01Click	Event will be fired when the user clicks on the OK button.

The event handling methods will be defined in the DynPage which we will create after we are finished with the JSP's.

When the user picked a number and clicks on the PLAY button (WelcomeMessage.jsp) we will display the picked number in an animated way and ask for his final confirmation. The JSP will look like that:



We use a **group** control with the headline "Portaloto - Player Informationt" and place the other controls in the group body. For a better placement of the controls we use the **gridlayout** control. The **gridlayout** allows us to center the buttons and text lines.

We create OutputNumberPicked.jsp. At the beginning of the JSP we have to do the following steps

- Import the resource package that we can access the resource to get the image
- Declaring the tagLib for HTMLB controls
- Declaring the bean which we need in the JSP (to get details about the scope of beans click here).

That looks like this:

```
<%-- import the resource package - will be used to get the image name --%>
<%@ page import="com.sapportals.portal.prt.resource.IResource" %>
<%@ taglib uri= "tagLib" prefix="hbj" %>
<%-- Declare Beans:
<%-- bean.StringBean contains two text strings that are displayed in textView T_Pick --%>
<jsp:useBean id="TextBean" scope="application" class="bean.StringBean" />
```

Then we open a content, a page and a form tag and start coding the HTMLB controls beginning with a **group** and **gridLayout**. The picture above shows the control and the control id we use in our JSP. In the JSP we code the controls "top - down" so with the picture it is very easy to follow the documented code in the OutputNumberPicked.jsp.

For the buttons we define events:

Control	Control id	Event	Event Handling Method	Description
button	OKFinal	onClick	onOKFinalClick	Event will be fired when the user clicks on the OK button.
button	Cancel01	onClick	onCancel01Click	Event will be fired when the user clicks on the CANCEL button.

The event handling methods will be defined in the DynPage which we will create now.

2. Setting up the Property File.

We create the property file Portaloto.properties. In the property file we define, besides the necessary entries (class name, HTMLB tagLib, service reference, JSP), self defined entries for values we access later in our DynPage. The self defined entries have the group name **MaxHits**. Please also have a look on the last line **ServicesReference** were we call the service HTMLB and com.sapportals.pdk.documentation.TutorialPortalotoService. com.sapportals.pdk.documentation.TutorialPortalotoService is a service supplied with the portal and manages the lottery.

The Portaloto.properties file looks like this:

```
ClassName=Portaloto
tagLib=/SERVICE/htmlb/taglib/htmlb.tld
MaxHits.description=Maximum numbers in the game
```

```
MaxHits.plaindescription=Maximum numbers in the game
MaxHits.type=Number
MaxHits.value=80
ServicesReference=htmlb, com.sapportals.pdk.documentation.TutorialPortalotoService
```

For more information about the property file, see "Property File"

3. Create the DynPage with an Initialization Part and the Event Handling Methods.

What a DynPage is and how we get one we already covered in the document "HTML-Business for Java - Building a DynPage". We now concentrate on the things important to our component.

We create a class MyDynPage in Portaloto.java. In the DynPage we have to override the methods **doInitialization, doProcessAfterInput** and **doProcessBeforeOutput.** In our component we use the methods as follows:

doInitialization method:
This method is called when the component starts. We access the property file and get the maximum number that can be picked with the according description ("Highest number in the game"). This information is displayed as textView in the WelcomeMessage.jsp. We use the highest number also in a for loop to fill the list box with entries - the lucky numbers. The lucky numbers start with 1 and end with the highest number from the property file. The list box entries are stored in an IListModel.

For the game the user has to be logged in. We use a method from the request object to find that out. If the user id is not set, the user is not logged in and we inform the user that he has to log in.

The information text strings and the IListModel are stored in beans and then we set the variable **JSPoutput** to `WELCOME_STATE` so that the method **doProcessBeforeOutput**, which is called immediately after this method **doInitialization**, calls the JSP **WelcomeMessage.jsp**.

doProcessAfterInput method:
This method is always called first whenever an event occurred. In this method we get the request, content and profile objects and recover the string bean. The event handling routines which are called immediately after this method **doProcessAfterInput** need the objects to have access to the controls in the JSP.

doProcessBeforeOutput method:
This method is always called last. We use a switch statement to call the JSP according to the **JSPoutput** variable. The **JSPoutput** variable is set by the **doInitialization** method and the event handling methods.

onSelect method:
This method is called when the user clicked into the list box to select a number. The method **getComponentByName** delivers the list box object which allows us to find out which number has been clicked with method **getMultiSelection**. The selected number is stored and put in a text string. The text string is displayed in the JSP to inform the user which number he picked. We set the variable **JSPoutput** to `WELCOME_STATE` so that the method **doProcessBeforeOutput**, which is called immediately after this method **onSelect**, calls the JSP **WelcomeMessage.jsp**

back to WelcomeMessage.jsp

onPlayClick method:
This method is called when the user clicked on the PLAY button. If the user is not logged in, the number is not accepted and we display an information message. Otherwise we set up an animation that displays the picked number in a graphical form. We split the picked number up in an upper and lower digit. The digits are then combined with the image name prefix. The prefix contains the sub folder of the resources (/images) and the filename SPHERE. As extension for the images we set ".gif". When we assume that the user selected the number 7 we have concatenated text string for `setName1="/images/SPHERE0.gif"` and for `setName2="/images/SPHERE7.gif"`. The two file names for the images are stored in the string bean. We set the variable **JSPoutput** to `OUTPUT_NUMBER_PICKED` so that the method **doProcessBeforeOutput**, which is called immediately after this method **onPlayClick**, calls the JSP **OutputNumberPicked.jsp.**

back to WelcomeMessage.jsp

onOK01Click method:
This method is called when the user clicked on the OK button in OutputText.jsp. OutputText.jsp, as described in the JSP section before,

serves as information window. So when the user clicked on OK he confirmed the message. We set up the text strings for the initial WelcomeMessage.jsp and set the variable **JSPoutput** to `WELCOME_STATE` so that the method **doProcessBeforeOutput**, which is called immediately after this method **onOK01Click**, calls the JSP **WelcomeMessage.jsp.**

back to OutputText.jsp

onOKFinalClick method:
This method is called when the user clicked on the OK button in OutputNumberPicked.jsp. OutputNumberPicked.jsp, as described in the JSP section before, displays the selected number in an animated form. When the user clicked on OK he finally confirmed the picked number. The picked number is stored in the com.sapportals.pdk.documentation.TutorialPortalotoService service. We set up the text strings for the initial OutputText.jsp to inform the user that his number has been accepted and when the number is drawn. We set the variable **JSPoutput** to `OUTPUT_TEXT` so that the method **doProcessBeforeOutput**, which is called immediately after this method **onOKFinalClick**, calls the JSP **OutputText.jsp.**

back to OutputNumberPicked.jsp

onCancel01Click method:
This method is called when the user clicked on the CANCEL button in OutputNumberPicked.jsp. OutputNumberPicked.jsp, as described in the JSP section before, displays the selected number in an animated form. So when the user clicked on CANCEL button we cease the process. We do that by leaving this method as empty shell.

back to OutputNumberPicked.jsp

onResultClick method:
This method is called when the user clicked on the RESULT button. We call the **getResult** method to get the result from the com.sapportals.pdk.documentation.TutorialPortalotoService service and set up the text strings for the information page. We set the variable **JSPoutput** to `OUTPUT_TEXT` so that the method **doProcessBeforeOutput**, which is called immediately after this method **onResultClick**, calls the JSP **OutputText.jsp.**

back to WelcomeMessage.jsp

4. Accessing the personalization data.

In the method **doInitialization** we access personal data. Personal data is stored in the property file. We use the file `Portaloto.properties` as property file. The content of the property file we described in section 2 above. We use the the `MaxHits.value` and `MaxHits.description`. We take the profile object and use the **getPropertyAttribute** method to get the associated value as string.

Example:

```
int maxHits = Integer.parseInt(profile.getPropertyAttribute("MaxHits", "value"));
String desc = profile.getPropertyAttribute("MaxHits", "description")
```

For more information about property files, see Accessing the Property file.

5. Accessing the com.sapportals.pdk.documentation.TutorialPortalotoService service

In the **Portaloto.properties** file we added the com.sapportals.pdk.documentation.TutorialPortalotoService service to the service reference. To use the service we added an `import` statement for **IStore** at the beginning of the Portaloto.java file and appended the methods.

```
// canPlay returns a true if you can play and false when not.
public boolean canPlay(IPortalComponentRequest request)

// Returns the result of the game (number drawn).
public String getResult(IPortalComponentRequest request)

// Returns the winner of the game.
public String getWinner(IPortalComponentRequest request)
```

```
// Gets the list of all games played today.
public Dictionary getGames(IPortalComponentRequest request)
```

6. Next steps

From here you can go to:

Internationalize the Portaloto component	Using resource bundles to make Portaloto multi lingual.

Tutorial Part II

The tutorial covers following topics:

- Creating resource bundles for English and German language.
- Changing JSP's and DynPage to access the resource bundles.

Click to run the example.

Prerequisites

This example uses the servlet class JSPDynPage from service HTML Business for Java. You must be familiar with this class and how the data is exchanged between the JSPDynPage and the JSP using a bean. This topic is covered by the document "HTML-Business for Java - Building a DynPage".
The locale settings (which language should be used by the browser and the portal) and how to change the settings are covered by the document "Internationalization"

Scenario

In Tutorial I we developed the lottery game for the portal. The user can select a number. This number is saved. A service "draws" a number and compares it with the selected number and notifies the winner. The messages in the lottery game should adjust to the selected language of the portal and/or the browser. To do that we use the Internationalization feature of the portal and create language files for english and german users.

Components for this scenario

Source File	Description
PortalotoInternational.java	Methods and Event handling DynPage.
Bean	**Description**
LuckyNumbersBean.java	Contains the selected number.
StringBean.java	Contains text strings that are displayed in the JSP.
JSP	**Description**
WelcomeMessage_int.jsp	Displays the introduction and a list box to select the number to play.
OutputNumberPicked_int.jsp	An animated confirmation of the selected number.
OutputText_int.jsp	Is used to display information or confirmation messages.
Property File	**Description**
PortalotoInternational.properties	PortalotoInternational property file. You must specify the services you are using in the components. It contains also information about the game.
Language Resource Files	**Description**
lotoResource.properties	Default language resource file. This file will be used, when no resource file exists for the language settings.
lotoResource_de.properties	Language resource file for language de (German)
lotoResource_en.properties	Language resource file for language en (English)

Setting up a JBuilder Project

If you have already developed a portal component you are familiar with the development cycle. If not have a look at the following

documentation:

Installation of the plugins	Set up JBuilder for a collaboration with the Portal Development Kit.
Guide	Developing with the JBuilder wizards.
Remote Debugging	Necessary steps to use the JBuilder debugging features to debug a portal component.

Changing the Source Code in the Servlet PortalotoInternational.java

In the com.sapportals.pdk.documentation.TutorialPortalotoService service so far we use text strings which are "hard coded" in the program.

Example:

```
beanst.setcell_row(1,"Pick a number and press the PLAY button.");
```

To use resource bundles we replace the text strings by a call to the resource bundle with a key that stands for the text string. To get the string from the resource bundle we use the method `resource.getString("key")`. "key" has to be defined in the resource file with the text string that is displayed. We use "speaking names" for "key" for easier use. To demonstrate that we look at a part of the code in the `doInitialization` method which uses four text strings to inform the logged in user that he can play and the not logged in user to log in first.

Code example from the `doInitialization` method as it is now:

```
if (request.getUser().getUserId() != null)
{
logged_in = true;
beanst.setcell_row(1,"Pick a number and press the PLAY button.");
beanst.setcell_row(2,"Numbers are drawn every evening at 6:00 o'clock.");
}
else
{
logged_in = false;
beanst.setcell_row(1,"Please log in first and then pick a number");
beanst.setcell_row(2,"and press the PLAY button.");
}
```

To use the resource bundles we need the resource object. At the beginning of the `doInitialization` method we got the resource object with:

```
request = (IPortalComponentRequest) this.getRequest();
context = request.getComponentContext();
profile = context.getProfile();
resource = request.getResourceBundle();
```

Now we change the lines as follows:

```
if (request.getUser().getUserId() != null)
{
logged_in = true;
beanst.setcell_row(1,resource.getString("Pick_a_number"));
beanst.setcell_row(2,resource.getString("Numbers_are_drawn"));
}
else
{
logged_in = false;
beanst.setcell_row(1,resource.getString("Please_log_in_first"));
beanst.setcell_row(2,resource.getString("Click_PLAY"));
}
```

Creating the resource bundles

We will provide an English and German version for the com.sapportals.pdk.documentation.TutorialPortalotoService service. We create the `lotoResource.properties` file first. This is the default resource file which is used when the portal or the browser is set to a language we do not support. As default language we use English. The `lotoResource.properties` is a text file which we can create in our JBuilder (or other IDE).

The content for `lotoResource.properties` for the four lines above is:

```
Pick_a_number=Pick a number and press the PLAY button.
Numbers_are_drawn=Numbers are drawn every evening at 6:00 o'clock.
Please_log_in_first=Please log in first and then pick a number
Click_PLAY=and click the PLAY button.
```

We can copy the `lotoResource.properties` file into `lotoResource_en.properties`. `lotoResource_en.properties` is the resource file which is used when the portal or the browser is set to the English language. To support the German language we create the `lotoResource_de.properties` file that looks like that.

```
Pick_a_number=Zahl auswählen und auf SPIEL Taste klicken.
Numbers_are_drawn=Zahlen werden täglich um 18:00 gezogen.
Please_log_in_first=Sie müssen angemeldet sein, um eine Zahl zu ziehen.
Click_PLAY=Dann Zahl wählen und auf SPIEL Taste klicken.
```

The language resource files have to be completed with all text strings used in the portal component.

Feel free to create another language resource file with the language of your country.

Defining the Resource Bundle in the PortalotoInternational.properties file

That the portal component can find the resource bundle we have to add the line

```
ResourceBundleName=lotoResource
```

to our `PortalotoInternational.property` file. For information about location and naming conventions of the resource bundle in the document "Internationalization".

Using the Resource Bundle in the JSP

The JSP contains controls that have language dependent text. The text strings in the JSP's are supplied by the `StringBean`. The `StringBean` is set by the DynPage. You can use the same procedure for the text strings in the buttons. To demonstrate another way we use the resource bundle directly in the JSP.

In the `WelcomeMessage_int.jsp` file we find the PLAY button. With no language support the PLAY button is defined:

```
<hbj:button
id="B_Play"
width="100px"
text="Play"
onClick="onPlayClick"
tooltip="plays the number"
/>
```

As you can see the text attribute (which defines the text string that is displayed in the button itself) and the tooltip attribute (which defines the text string that is displayed when you move the mouse pointer over the button) are language dependent.

To use the resource bundles in the JSP we have to et the resource object first. We do that inside the `form` tag with the command line:

```
<% ResourceBundle res = componentRequest.getResourceBundle(); %>
```

`res` is now the resource object. The PLAY button is defined:

```
<hbj:button
id="B_Play"
width="100px"
text="Play"
onClick="onPlayClick"
tooltip="plays the number" >
<%
B_Play.setText(res.getString("BUTTON.PLAY"));
B_Play.setTooltip(res.getString("TOOLTIP.PLAYS_NUMBER"));
%>
</hbj:button>
```

The keys BUTTON.PLAY and TOOLTIP.PLAYS_NUMBER have to be defined in the language resource files as well.

Example for `lotoResource_en.properties`:

```
BUTTON.PLAY=Play
TOOLTIP.PLAYS_NUMBER=plays the number
```

Example for `lotoResource_de.properties`:

```
BUTTON.PLAY=Spiel
TOOLTIP.PLAYS_NUMBER=Spielt die Zahl
```

Testing the Portal Component

Usually the portal has a global and user specific language setting. As long as there such settings the language settings of the browser are omitted. We recommend to switch off the portal language settings so that the browser language settings are taken. This allows easier testing of the portal component. For information about changing the portal language settings, see the document "Internationalization".

Index

- Known Bugs
- Frequently Asked Questions
  - Installation
  - License Key
  - Development
  - Error Messages - Tomcat/Portal
  - Error Messages - Portal Components
  - Knowledge Management

Known Bugs

Problem: When WINNT is installed on other drive than `C:` (e.g. `D:`) then an error occurs!!!
Message: `Copying pse to file C:\winnt\system32\verify.pse`
Workaround: Let system copy pse file to e.g. `C:\winnt\system32\verify.pse` and then copy it manually to e.g. `D:\winnt\system32\verify.pse`

Problem: After installing PDK the Client Side Eventing (EPCF) does not work in the PDK, but still works in EP 5.0 of same version.
Message: none
Workaround: You must specify a full qualified host name in the url (including domain, e.g. http://myhost.my.domain.com/...). http://localhost does no longer work for EPCF. You may also do a light modification to EPCF script according to CSN note 509499.

```
======================================================================
              Attachment for OSS Note #509499
          Retrofit of the corresponding code from 5.0.3.1
======================================================================
```

Step 1) In the directory where the portal is deployed <...>
you have to find the following java Script include

<...>\webapps\irj\services\epcfloader\script\standard\js13_epcf.js

Step 2) Back up this file

Step 3) Replace the block of the "old code" with the "new code"

<<<<<<<<<<<<<<<<<<<<<<< "old code" block begin >>>>>>>>>>>>>>>>>

```
EPCM.DB1.getMultiCookieDomain = function(){
// Other browsers allows cross domain access !!
return document.domain;
}
```

<<<<<<<<<<<<<<<<<<<<<<< "old code" block end >>>>>>>>>>>>>>>>>

<<<<<<<<<<<<<<<<<<<<<<< "new code" block begin >>>>>>>>>>>>>>>>>

```
EPCM.DB1.getMultiCookieDomain = function(){
// Other browsers allows cross domain access !!
return (document.domain.indexOf(".")<0) ? null : document.domain;
}
```

<<<<<<<<<<<<<<<<<<<<<<< "new code" block end >>>>>>>>>>>>>>>>>>

Frequently Asked Questions

Installation

Question:

I want to use Tomcat 4.0 with PDK

Answer:

1. Install JDK 1.3.1: http://java.sun.com/j2se/
2. Download Tomcat: http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release/v4.0.3/ and install Tomcat on your PC. We recommend to use `C:\tomcat` as target folder. Make sure that after the installation the %CATALINA_HOME% environment variable is set to `C:\tomcat`.
3. Create a temp folder under %CATALINA_HOME% (e.g. c:\tomcat\temp).
4. Download PDK: http://www.iviewstudio.com/ .
5. Extract PDK in the %CATALINA_HOME% folder.
6. Move all files from folder %CATALINA_HOME%/lib/apps to folder %CATALINA_HOME%/common/lib
7. Remove the files `tomcat.bat` and `restart.bat` from the folder %CATALINA_HOME%/bin (`tomcat.bat` is renamed to `catalina.bat` in Tomcat 4.0) .
8. Copy file servlet.jar from folder %CATALINA_HOME%/common/lib to folder %CATALINA_HOME%/webapps/irj/WEB-INF/lib/ext
9. Modify file catalina.bat in folder %CATALINA_HOME%/bin. Add the following lines under :okHome:

   rem --- Start SAP Portals Modification

   rem --- Insert your Proxy Settings here
   set _PROXY_HOST=proxy.mycompany.com
   set _PROXY_PORT=8080

   rem --- Set CATALINA_OPTS
   set CATALINA_OPTS=-Djava.library.path=%CATALINA_HOME%\common\lib -Dhttp.proxyHost=%_PROXY_HOST% -Dhttp.proxyPort=%_PROXY_PORT% -Dhttp.nonProxyHosts=localhost -Dcm.decode=X
   echo Using CATALINA_OPTS: %CATALINA_OPTS%

   set PATH=%CATALINA_HOME%\common\lib;%PATH%
   echo Using PATH: %PATH%
   rem --- End SAP Portals Modification

   If you receive a lot of error message during the first startup you did not create the temp folder under %CATALINA_HOME% the cause of the error messages. The PDK will work regardless of the error

Question:

I want to start Tomcat, but I get an error message:

```
Exception in thread "main" java.lang.NoClassDefFoundError: files\tomcat/lib/apps
```

Answer:

Please do not install Tomcat in a directory that contains blanks (like c:\program files). Move the complete tomcat folder to c:\ and set the environment variable to the new location.

Question:

I want to install a new version of PDK. What do I have to do?

Answer:

You must remove (do NOT only rename) the folder TOMCAT_HOME%/webapps/irj or move it to another location (e.g. C:/temp/irj ...). Then you follow the PDK installation guide.
Be aware, if you remove the folder TOMCAT_HOME%/webapps/irj all changes you have made in the old installation are lost!!!

Question:

I get a lot of error messages in the tomcat console.

Answer:

You have renamed a previous installation in the %TOMCAT_HOME%/webapps folder. You must delete or remove this previous installation from %TOMCAT_HOME%/webapps and the folder %TOMCAT_HOME%/webapps/irj.
Then restart tomcat. Now you should be able to start the portal via http://localhost:8080/irj/servlet/prt/portal

Question:

PDK doesn't startup after installation.

Answer:

Make sure you have used an empty tomcat installation. If you have reused an tomcat installation from a previous PDK installation make sure, that no old PDK libraries are stored in any of the tomcat directories.

Question:

Can PDK and Enterprise Portal be installed on the same machine?

Answer:

It can be installed on the same machine, BUT it can't run together on the same machine. You will receive some "strange"errors when you try to run a version on Tomcat (the PDK) parallel to JRun (EP).

Question:

Is the PDK particular about versions of Tomcat/ant/jdk/jbuilder?

Answer:

With JDK we need the same version like the Enterprise Portal, because in case of errors we must be sure that it is not because of an version problem of JDK. Of course you can use a newer version, but on your own risk. Ant and JBuilder can be used in any version you want to (as long the open tools are supported for JBuilder). We don't really need ant or JBuilder for the PDK.

Question:

Which edition of Jbuilder should I use- Personal, Professional, Enterprise?

Answer:

As long as the edition(personal, prof, enterprise) supports open tools, its fine You don't need any particular JBuilder edition. There are some restrictions in the personal edition (Remote debugging).

Question:

Why do we need to set Tomcat_proxy?

Answer:

Tomcat_proxy works like the proxy setting in your browser. If you don't need the setting to access a web page in your browser you don't need it in the tomcat as well.

Question:

Deploying with Jbuilder: Where does it deploy- Local or PCD?

Answer:

The Jbuilder open tool will deploy the par file in both directories (/deployment and /deployment/PCD). So you should not need the property localmodeallowed=true which you have to set if the .par file is only deployed locally.

License Key

Question:

After obtaining a (temporary) license we still get the message

```
"You do not have a valid license for the Enterprise Portal."
```

Repeating the procedure to obtain a temporary license doesn't succeed:

```
"A temporary license for this product already exists.
        You can't install another temporary license for Portal
        It's not possible to install the temporary license."
```

What can I do?

Answer:

First of all you have to restart Tomcat after installing the license.
If this doesn't help the current hardware configuration could have changed from the configuration you had when you initially installed the license key.
Changes of the hardware configuration could be:

- network connection active (applies for notebook installation, connection via Modem)
- external drives
- etc.

Please make sure that your hardware configuration is always the same as at the time when you initially installed the license.
Otherwise you have to reinstall your PDK (it is sufficient to delete the folder %TOMCAT_HOME%/webapps/ and start tomcat once) and to reinstall your license.
Note: There is a new licensing policy coming with version 5.0.2.0. See answer to next question.

Question:

Is applying a permanent license for PDK the same procedure as for the Enterprise Portal?

Answer:

This is the same procedure, but we will have a new license model with the next versions of PDK (5.0.2.0 and 5.0.3.0).

With version (5.0.3.0) you won't need a license key any longer if you are running the PDK installation only with 3 concurrent users ("anonymous user", "user#1", "user#2"). As additional named users access the same PDK installation they will get a message stating '*there is no valid license*'.

For the version 5.0.2.0 this will work similar (also without applying a temporary license), but there may be some side effects depending on your JBuilder settings, browser settings and such that can lead to the message that *there is no valid license* more often. So we recommend to still apply a temporary license for this version. The temporary license is easy to install:

1. Login to your local portal (this is now for version 5.0.2.0 possible without having a license!)
2. Start the license component (http://localhost:8080/irj/servlet/prt/portal/prtroot/LicenseComponent.default)
3. Press the button: *Install Temporary License*
4. *Restart Tomcat*

By the way: This is also the best way to install a permanent license.

Development

Question:

What is the portal data viewer?

Answer:

The Portal Data Viewer (PDV) is based on HTMLB. It is a convenient way to show tabular data. On top of the TableView Control of HTMLB the PDV offers the standard functionality of scrolling in the table but also some enhanced functions like personalization of the table (which column the user wants to see), searching, ...
There is a generic portal component of the Portal Data Viewer which is configured by a profile in order to display the data from a data source. Processing the data source and creating/filling the tableview (HTMLB) is performed by the PDV. If you need additional buttons, input fields, etc. in your portal component, or if you have to manipulate the data before output, you can use the PDV api. We recommend to extend the PDVDynPage. The PDVDynPage is an extension of the DynPage and provides a lot of functionality like handling of data source, personalization etc.

Question:

There are an example that comes with PDK on Basic Services Section (Internationalization).
When I run the example and change the language nothing happens. Why?

Answer:

Locate the file workplace.properties under
C:\tomcat\webapps\irj\WEB-INF\plugins\portal\system\properties.
Look for
request.mandatorylanguage=en
Change it to none.
request.mandatorylanguage=
Now restart tomcat and run the example again.

You are able to check your program with different languages, by changing the language in your browser settings. Via Tools ->Internet Options ->Languages you can choose a new language. This language setting is relevant to get the resource bundle.

In the general the following precedence is taken
1. Locale of the component (if specified in the profile [ForcedRequestLanguage, ForcedRequestCountry], potentially personalizable)
2. Mandatory Locale of the Portals (if specified in prtDefaults.properties)
3. Locale of the user (if available). The locale of the user is set in the file KMUsers.properties in the folder %IRJ_HOME%\WEB-INF\plugins\portal\services\usermanagement\data
4. Locale of the client (if send [i.e. if set up in (e.g.) IE under Tools/Internet Options/Languages...])
5. System default. The system default language is the VM system language. This can be specified by VM command line properties (-Duser.language=... and -Duser.region=.. ..) or (weaker) by setting the "Regional Options" on the "Control Panel" under Win2000.

Error Messages - Tomcat/Portal

Question

After installing the PDK I get following exception when starting up Tomcat.

```
ServiceManagerStartupException: keystore
.
.
```

Answer:

Check if the environment variable TOMCAT_HOME is set and points to the Tomcat folder .

Error Messages - Portal Components

New Portal Component - deployed for the first time

Question

"Could not instantiate implementation class MyClass of Portal Component My.default because: Could not find implementation class" - Exception occurs after the Portal Component has been deployed you try to execute it.

Answer:

The exception indicates that the portal can not find the class that is referenced in the property file default.properties with the property `ClassName` .
Make sure that the `ClassName` is spelled correctly (case sensitive) and if the class exists in the PAR file with the extension `.class`. You can open the PAR file with Winzip or another program capable reading ZIP files. The class referenced must be in the folder:

```
private/classes/...
```
or in
```
private/lib/MyClass.jar
```

The JAR file is also in ZIP format and can be viewed with the same program you opened the PAR file with.

Question

"Could not instantiate implementation class MyClass of Portal Component My.default because: Linkage error while loading implementation class"- Exception occurs after the Portal Component has been deployed you try to execute it.

Answer:

The exception indicates that it cannot find a service. Usually it is caused by an error in the default.properties file in the ServicesReference entry.

Correct entry:

ServicesReference=htmlb, Service2, MyService ....

Portal Component - redeployed

Question

Portal Component is deployed and executed correctly the first time. On any subsequent deployment you receive an error message "Could not instantiate implementation class xxx because: Could not find implementation class". Restarting tomcat solves the problem.

Answer:

If you are working with resource bundles (localization) this error occurs when the resource bundles are placed in the jar-file. Please remove the resource bundles from the jar-file. Place the resource bundles under the `/classes` or `/src` folder in your JBuilder project so that the PAR Wizard creates the par file properly.

Portal Component - Run time errors

Question

Null pointer error when accessing an inputField with type INTEGER.

Answer:

The inputField with type INTEGER is not `null`save. The error occurs every time the inputField is empty. You can use type STRING as work around.

Errors related with JSP and HTMLB:

Errors in JSP programs are mainly caused by

- case sensitivity has been ignored
- missing entry of the htmlb service and the the taglib declaration in the default.properties file
- mistyped attributes
- required attributes are not defined (e.g. id)
- tags are not properly closed
- quotes are missing or not closed

Errors in a JSP program are usually reported with a detailed description and a line number. In rare cases a general error message without a line number is generated. A short summary of error messages:

Error	example	errormessage
Missing ServiceReference entry in default.properties file.	ServicesReference=htmlb	Could not instantiate implementation class com.sapportals.wsc.examples.MyClass of Portal Component com.sapportals.wsc.examples.MyClass.default because: Linkage error while loading implementation class
Mistyped JSP header. "tagLib" is typed as "taglib"	<%@ taglib uri= "taglib" prefix="hbj" %>	ParseException: ID17195: Error in parsing taglib tag in web.xml or .tld file of the taglib library.
Missing entry in default.properties file for tagLib and/or ServicesReference.	tagLib=/SERVICE/htmlb/taglib/htmlb.tld ServicesReference=htmlb	Error in parsing taglib 'tagLib' tag in web.xml or .tld file of the taglib library.
This error usually occurs when a JSP becomes very big. JSP is precompiled in Java code. Then compiled with the Java compiler. The error is caused by if blocks which targets are further then 32kB away. Only solution so far is to make the JSP smaller		Exception caught : Error in service call of Resource Component : myComp.V4_Meldung.default Component class : null User : xxxxxx Original exception caught : (class: pagelet/_sapportalsjsp_editV4, method: doContent signature: (L.../IPortalComponentRequest;L.../IPortalComponentResponse;)V) Illegal target of jump or branch java.lang.VerifyError: (class: pagelet/_sapportalsjsp_editV4, method: doContent signature: (L.../IPortalComponentRequest;L...t/IPortalComponentResponse;)V) Illegal target of jump or branch
Required attribute missing	<hbj:label id="lab" width="300" />	ParseException: ID17214: Attribute id is required but was not found in the tag of the action. Error in file:myFile.jsp at line:21 position:7 Note: In this example the required attribute is "labelFor"
Using an id in a script that is not defined	success_message.setText ("The force is with you ");	Undefined variable or class name: success_message
Tag not properly closed	<hbj:textView id="t_v" <% t_v.setText("hallo" %> </hbj:textView>	ParseException: ID17121: Expecting '=' instead of '<'. Error in file:myFile.jsp at line:12 position:9
Missing quote	text="hallo	ParseException: ID17121: Expecting '=' instead of "". Error in file:myFile.jsp at line:10 position:24

Knowledge Management

Question:

When using knowledgemanagement (km) service in the PDK you may see a message

The page cannot be displayed
with a strange url looking like
`http://@cm.server.port@/irj/servlet/prt/...`

Answer:

You must specify the name of the servlet engine host within the configuration file

```
%TOMCAT_HOME%\webapps\irj\WEB-
INF\plugins\portal\services\knowledgemanagement\lib\config_local.properties
```

to override the settings in the configuration file `config.properties` at the same location.

Example for `config_local.properties`:

```
##########################################################################################
#02 Hostname of the local server
##########################################################################################
hostname=localhost:8080

##########################################################################################
# #03 Address of the database server
##########################################################################################
dbserver=localhost
```

Question:

When using knowledgemanagement (km) service in the PDK and trying to show the **details** of a file in the folder view the following message may appear:

```
Resource Not Found
/documents/New%c3%9fFolder/New%20Text.txt
The resource you are attempting to access is not available.
Check that the name or link is correct.
You might also check whether the respective repository is currently accessible
```

Answer:

The folder name or the file name may contain special characters (space, ä, Ä, ... ß, ...). On some servlet engindes (e.g. tomcat) the special characters are not decoded automatically. You may enforce the decoding (within the knowledgemanagement component) by passing the parameter "**cm.decode=X**" on to the knowledgemanagement component. Simply add to your java vm startup command:

java ... -D**cm.decode=X ...**

For Tomcat you should do this within `startup.bat` by changing the setting of TOMCAT_OPTS by appending this parameter:

```
set TOMCAT_OPTS=-Djava.library.path=%TOMCAT_HOME%/lib/apps ... -Dcm.decode=X
```