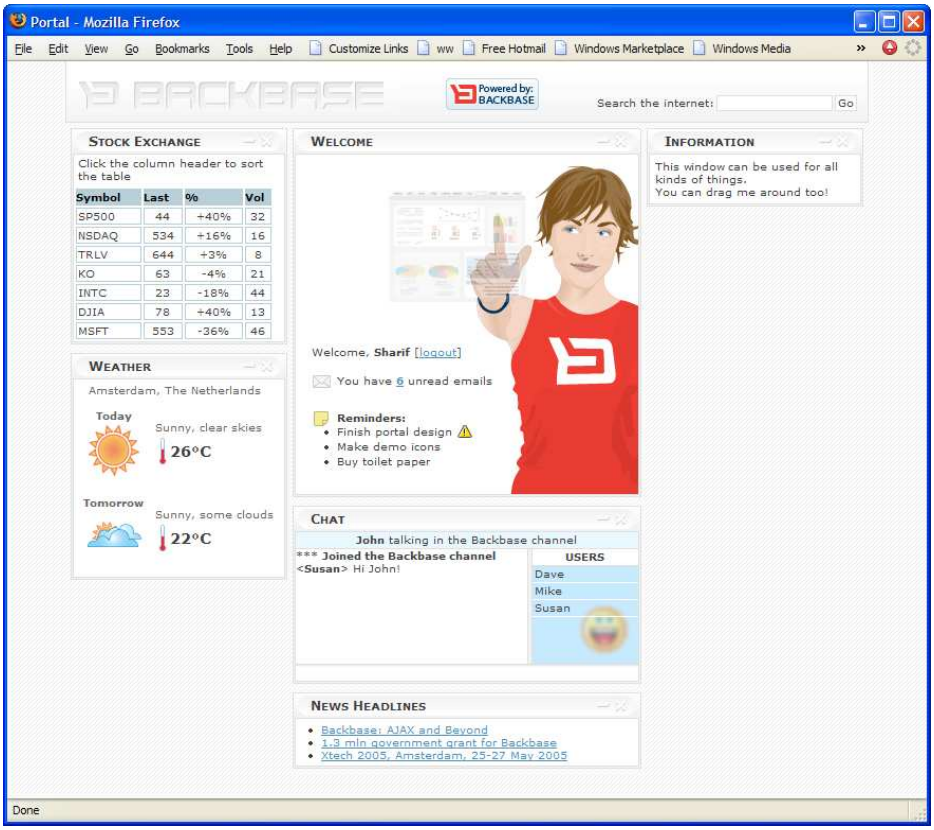
The Portal Starter Kit

1. Introduction

The Portal Starter Kit aims to illustrate the functionality that is required to build a portal application in BXML. It consists of the typical portal grid structure where portlets are shown in a three-column page layout.



The purpose of this document is to explain how such an application is built in BXML. It aims to briefly introduce and explain the key concepts that are important in the portal application. It is however not intended to be a complete or

final reference for any of the BXML functionality described here. Therefore you are advised to consult the main BXML Reference pdf (which should be located in the same directory as this Starter Kit Manual) for more detailed information about any functionality that you feel isn't explained clearly or satisfactorily here.

It is interesting to compare the Portal Starter Kit with the Windows Starter Kit. Both starter kits essentially contain the same content but in a different layout. The key difference is that content is shown in portlets in the Portal Starter Kit, and it is shown in windows in the Windows Starter Kit. The fixed three-column grid layout of the portal is replaced by a free-flow window-positioning layout.

2. The Core Structure of the Portal Application

The portal application can essential be broken up into 2 distinct functional regions. These regions in turn contain different modules.

This section will firstly give a brief functional description of these regions and their modules and then go on to explain the techniques used for determining the layout of these modules on the screen, for keeping the code for functionally distinct modules separate and for loading and unloading of the various modules.

2.1 The Regions

Portal Columns / Adding Portlets
The portal page layout has three columns. On the top of each column there is a menu to add new portlets to each column. This consists of a set of infoboxes, which is a BXML widget, which will be explained in more detail later. Since this site is only a demo, none of the links are actually functional.

The Portlets
Portlets are containers for arbitrary content. The portlets can be added or removed from the portal page, they can be opened and closed, and they can be moved between columns.

2.2 Include files

A key technique used for keeping functionally distinct modules separate from each other is the use of include files. Include files are well-formed XML files which contain both BXML and normal HTML. They can be small and simple, merely containing a few behavioral instructions or a small module such as a shopping cart. They can also be very large and themselves contain multiple nested include files.

In the portal application there are six files included from index.html:

Initially the basic portal functionality – defined as behaviors and custom controls – is included using the following statement:

```
<s:include b:url="portal.xml" />
```

Then the contents of each portlet window is loaded as an individual file:

```
<s:include b:url="stock_exchange.xml"/>
<s:include b:url="weather.xml"/>
<s:include b:url="welcome.xml"/>
<s:include b:url="chat.xml"/>
<s:include b:url="news.xml"/>
```

3. Event Handling and Behaviors

One of the key strengths of BXML is the ease and simplicity with which events can be handled. A unique feature of BXML is the behavior construct. A behavior is a generic construct in which, you the developer, can define which instructions or in BXML terminology tasks should be executed when a given event occurs. This behavior can then be applied to any given element, which will then inherit all of the event handlers defined within the behavior. This makes it easy to reuse functionality and also to separate the structure of the page from the behavior.

The following portal-search-text behavior is a relatively simple behavior, which is used by the input box at the top of page, which is used to search the internet with. You can find it in the portal.xml include file.

```
<s:behavior b:name="portal-search-text">
  <s:state b:on="deselect" b:normal="portal-search-text"/>
  <s:event b:on="blur">
    <s:task b:action="hide" b:target="id('portal-search-error')"/>
  </s:event>
</s:behavior>
```

As you can see, this behavior contains two child elements, one s:state and one s:event element. Both of these tags subscribe to a particular event and form an event handler for it. The s:event tags are the most common children of behaviors where as s:state tags are a special case, so lets look at the s:event first. It is triggered by a blur event, which is the event that goes off when the input box loses its focus. This event handler contains a single s:task tag, which contains a hide action. A hide action makes an element on the screen invisible. This action needs a target, which is of course given by the b:target attribute. This attribute contains the following XPath statement:

```
id('portal-search-error')
```

This relatively simple XPath, searches for the element with the id attribute of `portal-search-error'. Once this element is found, the hide action will proceed to make it invisible.

The s:state tag is a specialised form of s:task, which is used solely for enabling CSS class changes to be linked to state changes and mouse movements. It makes it very easy to make attractive mouse roll-overs. This s:state tag is only active when the input box is deselected, which is its default state. It can define up to 3 different sets of classes to be used by this button depending on whether it is in its normal state, or whether the mouse is hovering over it, or whether the mouse button is pressed down. These are defined by the b:normal, b:hover and b:press respectively. In this case it only defines a class for the normal case.

4. Portal Controls

An important technique used in building BXML application is the creation and reuse of custom client controls. These are also sometimes referred to as widgets. Basically one of these custom controls is a definition for a new BXML tag, which can then be used throughout your application.

Although the controls are fairly simple in most cases of the Windows Starter Kit, it is good practices to place reusable UI elements in a control definition. Lets examine the way windows are built up in this application.

```
<b:portlet id="portlet-stockexchage">
  <b:portlet-head b:caption="Stock Exchange"/>
  <b:portlet-body>
    <s:include b:url="stock_exchange.xml"/>
  </b:portlet-body>
</b:portlet>
```

As you can see essentially the portlet is built up out of 4 main tags. The b:portlet tag is the main container, this contains a b:portlet-head tag and a b:portlet-body tag. The b:portlet-body tag finally contains an s:include tag which links to an external file, which contains the real body of the portlet. Both the b:portlet tag, the b:portlet-head tag and the b:portlet-body tag are custom tags, which are defined not by Backbase, but elsewhere in the application.

So lets take a look at one of these definitions and see how these tags are translated into portlets, you can find all of these definitions by searching through the portlets.xml file.

```
<s:htmlstructure b:name="b:portlet">
  <div>
    <div class="b-portlet-inner">
      <s:innercontent/>
    </div>
  </div>
</s:htmlstructure>
```

As you can see the definition for this control consists of a special tag called s:htmlstructure. This is the tag, which is used to define new tags. With the b:name attribute you give the name of the new tag, which is b:portlet in this case. All custom defined tags must be in the b namespace. Within the s:htmlstructure tag you can insert any HTML tags which you want to be rendered when the new tag is used. Note that only HTML tags may be used within the s:htmlstructure tag. Finally inside this HTML the s:innercontent tag is inserted at the point where you want the child elements of the new tags to be placed. Obviously the child elements of the new tag may be any BXML tag and not just HTML. When we examine the HTML that makes up this new b:portlet tag, it doesn't seem to do much more than place two div tags around the contents. The inner of these two div tags has a class applied to it. This class is a normal CSS class, which follows the normal CSS rules that any other page in a particular browser will follow. If you want to you can examine it in portal.css.

So all in all b:portlet doesn't seem to do much that is very useful, but there is more. A window behavior has also been defined and an s:default tag is also specified. This portlet behavior is large and complicated and it will not be shown here in full. Instead parts of it will be explained as and when necessary. Before

we go on it should be noted that it is the s:default tag, which is used to bind this behavior to all instances of b:portlet.

```
<s:default b:attribute="b:behavior" b:value="portlet" b:tag="b:portlet" />
```

The s:default tag is relatively easy and simple to understand. It simply binds a given value to a given attribute on a given element. In this case it is the portlet behavior which gets bound to all b:portlet elements. Now by examining some key parts of this portlet behavior the working of the b:portlet element should become clear.

```
<s:event b:on="init">
  <s:choose>
    <s:when b:test="@b:disabled='true'">
      <s:task b:action="trigger" b:event="disable"/>
    </s:when>
    <s:otherwise>
      <s:task b:action="trigger" b:event="enable"/>
    </s:otherwise>
  </s:choose>
</s:event>
```

The code fragment above is from the event handler for the init event. This is a custom event, which is triggered by the construct event of portlet. This event gets triggered when, a new portlet has been rendered. This init event handler start with a s:choose command. This s:choose tag works similarly to the JavaScript switch operator. A number of tests are performed sequentially by the s:when tags until one of the tests returns true. If none of the tests return true, then any instructions in the s:otherwise tag gets executed. In this case the tests performs an XPath instruction, which retrieves the value of the b:disabled attribute and then compares this to a string value. If it is ‘true’ then it triggers a disable event. Since there is no b:target attribute an implicit target of the element self is used. If the value of the b:disabled attribute is not ‘true’ or the attribute is not present then the s:otherwise tag is executed, which triggers a enable event on the element.

Now lets take a look at the disable event handlers:

```
<s:event b:on="disable">
  <s:setatt b:disabled="true"/>
  <s:task b:action="hide"/>
</s:event>
```

This is a relatively simple event handler which executes a few tasks. It starts off with a s:setatt tag. This s:setatt instruction causes any attributes found within this tag to be set onto the actual element that uses this behavior. In this case the b:disabled tag gets set to ‘true’. This might seem a little strange, if you think that we already tested this in the previous event handler, but you shouldn’t forget that the init event handler doesn’t have to be the only place in which a disable event can be called. The next task is also relatively simple; it hides the portlet.

In a more or less similar way much of the rest of the portlet controls are built up. This document is in no way meant to provide a complete explanation of how the Portlet Starter Kit works or to cover every aspect of BXML. You are advised to look through the code of Portlet Starter Kit yourself and examine the rest of the BXML documentation where necessary.