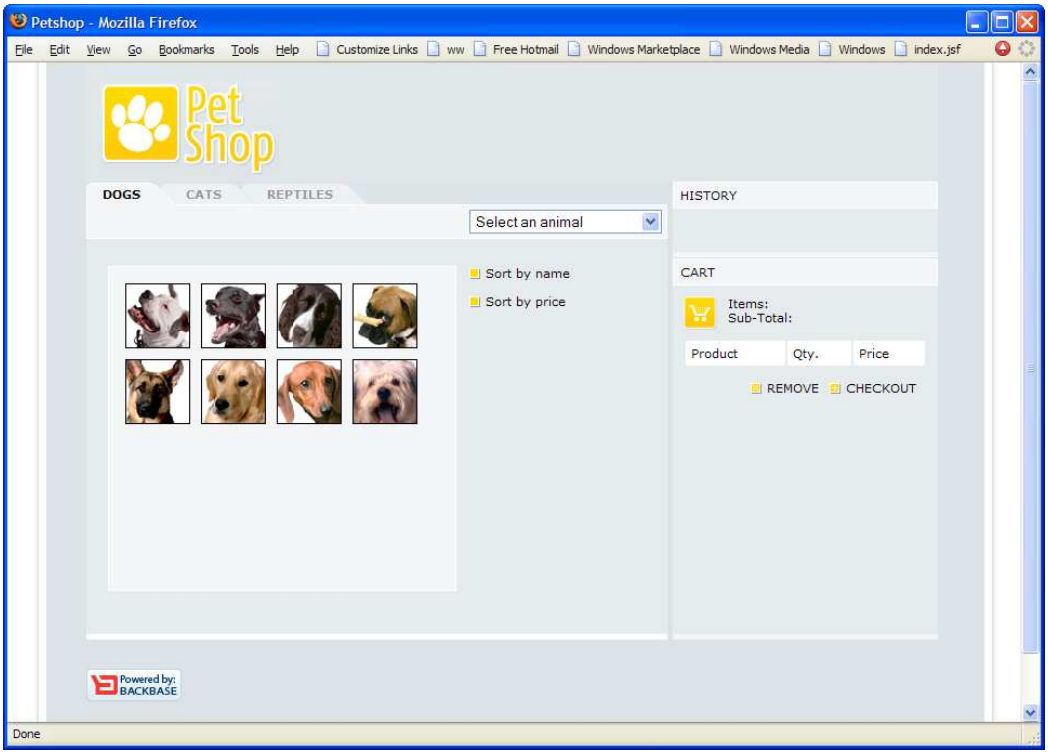
The Pet Shop Starter Kit

1. Introduction



The Pet Shop starter kit aims to illustrate the functionality that is required to build a basic shop type application in BXML. It consists of a relatively simple interface, which allows a user to go through the process of purchasing a pet.

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 Pet Shop 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.

2. The Core Structure of the Pet Shop Application

The Pet Shop application can essentially be broken up into 4 distinct modules. This section will firstly give a brief functional description of these 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 Modules

Display Area

This is where the user can view the available pets and find out more information about them. Pets are available in 5 different categories, like dogs and cats etc. It is possible to move from one category to another category by clicking on the appropriate tab. Within a category, the animals can be sorted by name and by price. A pet can be selected and more information about this type of pet is then displayed.

Shopping Cart

This area shows details of the pets that have been selected for purchased. It also adds up the current total cost and contains a link to the actual check out.

History

The history keeps track of all pets that a customer of the pet store has looked at during the shopping session. Clicking on the thumbnail of a pet in the history will take the customer back to the corresponding detail page.

Checkout

This area contains a form, which allows the user to enter his or her personal information that is required for shipping and billing of the purchased pets.

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 pet shop application several files are included right at the start of the main index.html file.

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

These two include statements are used to load in the code which is needed to enable the button and the tabbox elements which are used in the pet store. In BXML terminology buttons and tabboxes are called widgets.

2.3 Defining Screen Partitioning

The most useful BXML tool for controlling screen layout is the panelset. Panelsets work in much the same way that a frame set does. You can easily divide a screen up into rows and columns, which are called panels. These rows and columns have either a fixed width expressed in pixels, ems, picas etc., or they can have a relative width given as a percentage of the currently available width of the panelset parent. Alternatively a panel can fill up the remaining available space using the wildcard (*) sign. The key difference between a panelset and a frameset is that a panelset is entirely built using div elements and therefore does not consist of separate files like a frameset.

The main index.html page of the pet shop application is built up of a large nested panelset. The code below shows the outline of the panelset used to build up the index page.

```
<b:panelset b:cols="55px * 58px">
  <b:panel style="background: url('images/app/bg-left.gif')">
    <!-- left side background image -->
  </b:panel>
  <b:panelset b:rows="110px *">
    <b:panel style="background-color: #e1e6e9;">
      <!-- Pet store Logo -->
    </b:panel>
    <b:panelset b:cols="* 245px" style="background-color: #dde3e6;">
      <b:panel style="overflow: visible; padding-right: 5px;">
        <!--Main Pet Categories and information -->
      </b:panel>
      <b:panel class="b-panel">
        <!-- Shopping Cart and History area -->
      </b:panel>
    </b:panelset>
  </b:panelset>
  <b:panel style="background: url('images/app/bg-right.gif')">
    <!-- right side background image -->
  </b:panel>
</b:panelset>
```

Admittedly this is fairly complex. If you inspect it carefully you will see that it actually consists of 3 panelsets nested within each other. A panelset can simply contain panels as child elements but it can also contain another panelset, which in turn can contain more panels – usually in a different orientation.

The image below should help you make sense of the panelset above. It is a screenshot of the index page with a set of lines superimposed on it to indicate the panels and panelsets. The dotted lines indicate the orientation of the panelset i.e. whether it is made up of columns or rows and the size of the panels. As mentioned above the asterix (*) is used to indicate that the panel or the panelset, which it corresponds to, may use up the rest of the available space.

The different colors are used to indicate panels within each of the nested panelsets. Black corresponds to the outermost panelset, blue to the middle panelset and red to the innermost panelset.

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 used by 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 document from the behavior.

The behavior shown below is a relatively simple behavior that allows the element, which uses this behavior to be shown when it is selected and hidden when it becomes deselected.

```
<s:behavior b:name="show-hide">
  <s:event b:on="select">
    <s:task b:action="show"/>
  </s:event>
  <s:event b:on="deselect">
    <s:task b:action="hide"/>
  </s:event>
</s:behavior>
```

A behavior can be used by a given element by placing the b:behavior attribute on it. The show-hide behavior is applied to a panelset in the index.html file in the code fragment below.

```
<b:panelset b:behavior="show-hide" b:cols="120px *" style="display:
none;">
```

Event handlers don't have to be as simple as the ones above, but can instead contain multiple tasks or fxstyles. The behavior can also be made more

sophisticated by applying certain control flow tags like the s:if tag or the s:choose tag. The follow code fragment from the button.xml include file, which is used to store the functionality used for the b:button tag, demonstrates this:

```
<s:behavior b:name="b-button">
  <s:event b:on="construct">
    <s:choose>
      <s:when b:test="@b:icon = 'empty'">
        <s:task b:action="addclass" b:value="b-button-empty" />
      </s:when>
      <s:when b:test="@b:icon = 'previous'">
        <s:task b:action="addclass" b:value="b-button-previous" />
      </s:when>
      <s:when b:test="@b:icon = 'next'">
        <s:task b:action="addclass" b:value="b-button-next" />
      </s:when>
      <s:when b:test="@b:icon = 'remove'">
        <s:task b:action="addclass" b:value="b-button-remove" />
      </s:when>
      <s:otherwise></s:otherwise>
    </s:choose>
  </s:event>
  <s:event b:on="mousedown">
    <s:task b:action="addclass" b:value="b-button-down" />
  </s:event>
  <s:event b:on="mouseup">
    <s:task b:action="removeclass" b:value="b-button-down" />
  </s:event>
  <s:event b:on="mouseleave">
    <s:task b:action="removeclass" b:value="b-button-down" />
  </s:event>
</s:behavior>
```

As you can see the construct event of the b:button is given its own event handler. This construct event is triggered when either the index.html file is loaded or when an include file gets loaded in. When the construct event is handled the interpreter finds a s:choose tag. 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:icon attribute and then compares this to a string value. If it matches one of the string values, like ‘empty’ or ‘next’, then it performs an addclass task which appends an additional class to the b:button element.

The b-button behavior also defines a set of actions to be taken when the button is being clicked. Adding and removing a class during the mouseup and mousedown events graphically represent the clicking of the button to the user. It should be noted that any HTML or BXML element can have multiple CSS classes at any one time. Therefore adding or removing specific classes does not affect other ‘baseline’ classes that the element may have.

4. XPath

An essential aspect of BXML is the ability to target elements on the screen and to be able to retrieve information about these elements or their attributes. The XPath language is used for all of these types of operations. Almost every task that is executed has a target upon which this task should be executed. This target is not always visible in the statement, since the default target for most tasks is the element, which is using the behavior, itself. It is very important to have at least a basic understanding of XPath if you want to be able to build BXML applications. The more thoroughly you understand XPath, the more you will be able to leverage its power across BXML and the more powerful and flexible applications you will be able to create.

XPath statements which are used to select on screen elements are commonly found in the b:target attribute of a s:task statement or other statements with an action in it. For example if you examine the tag that is used to created the 'CHECKOUT' button at the bottom of the shopping cart, you can see that there is an inline action, which will be triggered by the command event that occurs when this button is clicked:

```
<b:button
	b:icon="next"
	b:action="select"
	b:target="id('checkout')">CHECKOUT</b:button>
```

Clicking on the button will therefore trigger a select action. The target of this selection is indicated by the XPath statement:

```
id('checkout')
```

This will cause the buffer with the id attribute of 'checkout' to be selected. This b:buffer is a special tag with a link to another XML include file, which isn't loaded in until it becomes selected. So by clicking on the button a trigger of events will occur, which cause the checkout.xml file to be loaded, allowing the user to complete his current purchase.

Now lets look at a somewhat more complicated XPath statement. In the cart-receiver behavior, which is used by the Shopping Cart part of the application and stored in the petshop.xml include file you can find the following task as part of the calculate event handler.

```
<s:task
	b:action="set"
	b:target="id('item-count')"
	b:attribute="innerHTML"
	b:value="{sum(id('cart')/b:item/b:item-quantity)}" />
```

Now when analysing an XPath there are two important things to be aware of. Firstly there is the question of data type. Different attributes expect the XPath resolver to return different types of values to them. A b:target for example expects an element or a collection of elements to be returned to it – this is called a nodeset in XPath parlance. A b:test expects a boolean value to be returned to it. And a b:value attribute on the other hand expects a string value to be returned to it. If the 'wrong' type of value gets given as the result of an XPath then the BXML engine will try to convert it to the expected data type. So in the case above the b:target again contains a fairly straight forward id based XPath. The b:value however is more interesting. Normally a b:value attribute expects a static string. However it is possible to make the content of the b:value dynamic, by using an XPath statement as its value. However since the default value for a b:value attribute is a static string, {curly braces} are placed around its value to indicate that the contents of the attribute are an XPath statement and should be resolved using the XPath parser.

Examining the XPath itself we see:

```
sum(id('cart')/b:item/b:item-quantity)
```

What this means is that all b:item-quantity children of the b:item elements in the cart should first be collected and then the sum of these elements should be returned. Since b:item-quantity elements are elements and not values, the XPath resolver automatically takes their text value and uses this in the computation.

5. Templating

A key facet of this starter kit is that the entire catalogue of pets and all of the information that is maintained about them are stored in a single XML file: catalog.xml. This isn't a BXML file, but simply a 'normal' XML file. Such a file can easily be updated or generated on the fly using information stored in the database. But since it is 'normal' XML and not BXML, before it can be gainfully used in the Pet Shop application it needs to be transformed. This transformation is done by the BXML templating mechanism. This templating mechanism is very similar to that of XSLT. An attempt has been made to use all of the same tags that are used in XSLT so that those of you that are familiar with XSLT should find it very easy to start using this templating mechanism. The transformation of the catalog.xml file is done according to the instructions contained within a stylesheet. The result of this transformation is then moved into the main tabbox, where it is displayed.

The command, which initiates this process, is the render command. A small div element at the bottom of the index.html file contains this instruction in its construct event handler.

```
<s:event b:on="construct">
  <s:task b:action="render"
          b:stylesheet="b:url('stylesheet.xml')"
          b:xmldatasource="b:url('catalog.xml')"
          b:destination="id('main-content')"
          b:mode="aslastchild"/>
</s:event>
```

Since this is the key action in this whole process lets look at each of the attributes and explain what their values mean.

- b:stylesheet: This attribute contains a special XPath statement, which targets the stylesheet. If a stylesheet is used that is part of a BXML file then the `b:xml('xxx')` syntax should be used, where the 'xxx' portion of the XPath statement is equal to the b:name attribute of the stylesheet. If on the other hand the stylesheet is stored in a separate XML file, as it is in this case then a syntax like `b:url('stylesheet.xml')` syntax should be used, where the `'stylesheet.xml'` portion of the XPath statement is equal to the to an absolute or relative path of the stylesheet file. In this case it is a relative path.
- b:xmldatasource: This attribute contains a special XPath statement, which targets the XML data source. If a XML data source is used that is part of a BXML file then the `b:xml('xxx')`syntax should be used, where the 'xxx' portion of the XPath statement is equal to the b:name attribute of the XML data source. If on the other hand the data source is stored in a separate XML file, as it is in this case then a syntax like `b:url('catalog.xml')` should be used, where the `'catalog.xml'` portion of the XPath statement is equal to the to an absolute or relative path of the data source file. In this case it is a relative path.
- b:destination: this attribute is another XPath statement, which is used to indicate the destination into which the result of the transformation should be placed. In this case it points toward the main tabbox, which shows the different categories of pets.
- b:mode: This attribute specifies how the result of the transformation should be placed into the destination. The default value is replace. However in this case the result is placed into the tabbox as its last child.

Please node that the b:url() and b:xml() XPath statements used for the values of b:stylesheet and b:xmldatasource are a custom extension to XPath. They are not

part of the W3C standard. This is why they have been prefixed with the Backbase namespace: b.

You are advised to examine both the catalog.xml file and the stylesheet.xml file carefully. It is beyond the scope of this introductory manual to explain the inner workings of XSLT or to even explain this transformation process in great detail. If you need to learn XSLT you should consult a good XSLT reference guide or developer site. However in the space that is left of this introduction, lets take a quick dip into this transformation and pick out the parts that are of special importance to BXML.

While examining the catalog.xml file, you can see that it has the following structure. There are categories of pets, such as dogs, cats and reptiles. These categories, contain products, such as bulldogs and boxers and finally the products contain items, which are usually males or females. The transformation process makes two separate type of views based on this information. The first is the one that is initially shown when one of the tabs is clicked. It is equal to all of the products within a single category – all dogs for example. The second view is shown when a product, such as a bull dog is selected. This view shows a detailed picture of this product, including all of its items. By looking at the stylesheet.xml you can see this structure, with two main views, within it.

The most important tag within such a style sheet is the s:template tag, which contains the rules to apply when a certain template is matched. There are 3 such s:template tags within stylesheet.xml.

```
<s:template b:match="/">
  <b:deck style="height: 400px;">
    <s:apply-templates b:select="categories/category"/>
  </b:deck>
</s:template>
```

The first s:template tag has a b:match attribute with the value of:

```
/
```

This is a very simple XPath which simple matches the root node of the XML data source, which is the categories element. It then inserts a b:deck tag. A b:deck tag is basically just a container. It can have div elements or some other type of elements or b:buffer elements as its child elements. What makes a deck different from a normal div is that only one of its child elements is ever visible at one point. The visible child is the selected child. The first child of the deck is always selected by default and therefore visible at start up. This deck is of key importance for the working of the tabbox, which needs to keep the selection of a particular tab synchronized with a particular piece of contents that represents this tab. If you are interested in exactly how a tabbox works and how it synchronizes its state information with the contents of the tabbox then you should consult the manual of the Vacation Starter Kit. This explains this whole process in great detail.

Once the b:deck has been inserted into the output of this transformation, then you see a s:apply-templates tag. This tag gives back control to the XPath Resolver and resolves the XPath statement in the b:select attribute.

```
categories/category
```

This selects all categories child nodes of the root and then selects their category children. It then goes and searches for another s:template tag which matches this selection:


```
<s:template b:match="category">

  ....

</s:template>
```

The HTML and the 'normal' BXML contents of this template then gets placed into the output for each category element that was found in the selectoin. The XSLT specific tags get executed, until another s:apply-templates tag is found and then the process starts again. This is in short how the templating mechanism uses the stylesheet to transform an XML data source.

Once again let us reiterate that this document is in no way meant to provide a complete explanation of how the Pet Shop Starter Kit works or to cover every aspect of BXML. You are advised to look through the code of Pet Shop Starter Kit yourself and examine the rest of the BXML documentation where necessary.