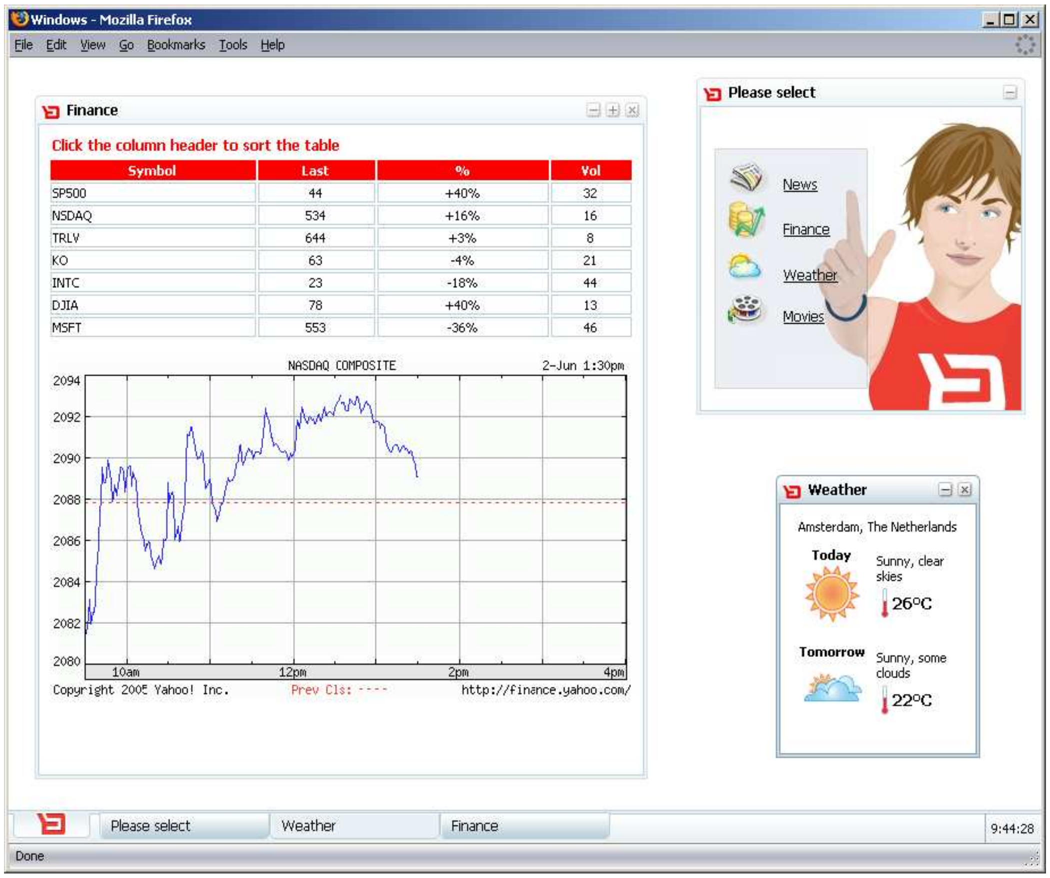
The Windows Starter Kit

1. Introduction

The Windows Starter Kit describes the functionality that is required to build a windows application in BXML. Its basic layout of the application is a free form desktop; you can open new windows from a menu, the windows can be resized and placed anywhere on the desktop, and you can minimize and maximize the windows.



Click the column header to sort the table

Symbol	Last	%	Vol
SP500	44	+40%	32
NSDAQ	534	+16%	16
TRLV	644	+3%	8
KO	63	-4%	21
INTC	23	-18%	44
DJIA	78	+40%	13
MSFT	553	-36%	46

Amsterdam, The Netherlands

Today: Sunny, clear skies, 26°C

Tomorrow: Sunny, some clouds, 22°C

It is interesting to compare the windows starter kit with the portal starter kit: both essentially contain the same content, but use a different layout. The key difference is that content is shown in portlets in the portal starter kit, and 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.

Although the purpose of this document is to explain how to build a Windows application using BXML, it is not intended to be a complete or definitive reference for any of the BXML functionality described. For detailed information on the functionality described, refer to the BXML Reference PDF.

2. The Core Structure of the Windows Application

The windows application can essential be broken up into 2 distinct functional regions, the Window Manager and the Windows, which in turn contain different modules.

This section will firstly provide 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

Window Manager
The window manager, a bar at the bottom of the application layout, controls the windows. From the window manager, new windows can be opened and minimized windows can be re-opened.

The Windows
Windows are containers for arbitrary content. Windows can be closed, minimized and maximized.

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, or they can be very large and themselves contain multiple nested include files. In the portal application four files are included from the main *index.html*:

Initially the basic window, windowarea and taskbar functionality – defined as behaviors and Backbase controls – is included with the following statements:

```
<s:include b:url="../../controls/backbase/b-taskbar/b-taskbar.xml" />
<s:include b:url="../../controls/backbase/b-window/b-window.xml" />
<s:include b:url="../../controls/backbase/b-windowarea/b-windowarea.xml" />
```

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 Window Starter Kit has a fairly simple page layout. There is only one panelset with two rows. The task bar at the bottom has a fixed height. The rest of the screen is dynamic in height and consumes the remaining screen space.

```
<!-- Main Panelset -->
<b:panelset b:rows="* 28px">
  <b:panel>
    <!-- Windowarea Panel -->
  </b:panel>
  <b:panel>
    <!-- Taskbar Panel -->
  </b:panel>
</b: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 applied to any given element, which then inherits all 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 *b-window-closebutton* behavior is a fairly simple behavior which illustrates the key structure of a behavior and two different features of behaviors – state control and task execution. You can find this behavior in the main *b-window.xml* include file. The behavior is used by the close buttons on all windows in the Window Manager.

```
<s:behavior b:name="b-window-closebutton" b:behavior="b-window-button">
  <s:initatt b:tooltiptext="Close" />
  <s:state b:on="deselect"
       b:normal="b-window-button b-window-closebutton"
       b:hover="b-window-button b-window-closebutton b-window-button-hov"
       b:press="b-window-button b-window-closebutton b-window-button-pres" />
  <s:event b:on="command" b:action="trigger" b:event="close" b:target="../../.." />
</s:behavior>
```

As you can see, this behavior contains two child elements; an *s:state* element and an *s:event* element. These elements define event handlers for when a particular event is triggered. The *s:event* tags are the most common children of behaviours, whereas *s:state* tags are a special case, so lets look at the *s:event* first. It is triggered by a *command* event, which is the event that is triggered when an item is clicked (mouse or keyboard). So when a user clicks the close button on a window, the *command* event is triggered and the *command* event handler is executed. Event handlers can contain *s:task* tags, which are used to execute actions. However, since this event handler only has one task, you can use the shorthand. The *trigger* action is defined on the event handler which is used to manually trigger an event on a target element. In this case, it triggers a custom event called *close*, which is triggered on the window to take care of closing the window. (The *b:target* attribute indicates that the target of the event is the window; to understand properly what the attribute is doing you first have to understand some XPath, which will be explained in section 4.)

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 close button is deselected, which is its default state. It defines three different sets of classes to be used by this button depending on:

- whether it is in its normal state
- whether the mouse is hovering over it
- whether the mouse button is pressed down.

These classes are defined using respectively the *b:normal*, *b:hover* and *b:press* attributes.

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 on which the task should be executed. The target is not always visible in the statement, since the default target is the element itself which is using the behavior (known as the "context node"). 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.

A simple example of an XPath statement is the targeting of an element based on its id attribute. To keep things simple, take the following example (not to be found in the starter kit):

```
<s:behavior b:name="simple-show-hide">
  <s:event b:on="select" b:action="show" b:target="id('popupWindow')" />
  <s:event b:on="deselect" b:action="hide" b:target="id('popupWindow')" />
</s:behavior>
```

When an element that uses this behavior becomes selected, a *show* action is executed on the element indicated by the b:target attribute which has the following value:

```
id('popupWindow')
```

This causes the element with the *id* attribute of 'popupWindow' to be used for the *show* action. On the *deselect* event of the element the inverse action will happen, which will hide the element with the *id* attribute of 'popupWindow'.

5. Window Controls

An important technique used in building BXML application is the creation and reuse of custom controls. 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 used in most of the Windows Starter Kit are fairly simple, it is good practices to place reusable UI elements in a control definition. Let's examine the way windows are built up in this application:

```
<b:window id="window_news" style="left:10px;top:10px;width:370px;height:290px;">
  <b:windowhead><!-- window title here --></b:windowhead>
  <b:windowbody>
       <!-- window content -->
  </b:windowbody>
</b:window>
```

As you can see, essentially the window is built using 3 main tags: The *b:window* tag is the main container, and this contains a *b:windowhead* to define the title of the window and a *b:windowbody* tag that contains the window content. These three tag are custom tags, which are defined not in the BPC, but elsewhere in the application.

So let's take a look at these definitions and see how these tags are translated into windows. You can find all of these definitions by searching the contents of the *b-window.xml* file:

```
<s:htmlstructure b:name="b:window" b:behavior="b-window">
  <div><s:innercontent /></div>
</s:htmlstructure>
```

As you can see, the definition for this control consists of a special tag called *s:htmlstructure*. This tag is used to define new tags; the *b:name* attribute is used to the tag a name, *b:window* in this case. 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:window* tag, it doesn't seem to do much more than place one *div* tag around the contents.

So all in all, the *b:window* doesn't seem to do much that is very useful, but there is more. A *window* behavior has also been defined. This *window* behavior is relatively complicated and it will not be described here in full; only a few parts of the code will be explained, when necessary. Before we continue, note that the *b:behavior* attribute can be used on a *s:htmlstructure* tag to bind this behavior to all instances of *b:window*. For example:

```
<s:htmlstructure b:name="b:window" b:behavior="b-window">
```

Now by examining some key parts of this *window* behavior the working of the *b:window* element should become clear.

One of the main event handlers in the behavior is for the *construct* event. This event is triggered as soon as the *b:window* tag is rendered. Within this event handler, a large number of instructions are contained for setting up windows and where, depending on the attributes on the *b:window* tag, different decisions will be made about the final structure of the window.

One example of this is the placement of “window buttons” to be able to use the *close, minimize* and *maximize* functionality of the window.

```
<s:if b:test="not(@b:windowbuttons = 'none')">
   <s:render b:destination="b:windowhead" b:mode="aslastchild">
      <div class="b-windowbutton-container"></div>
   </s:render>
   <s:variable b:name="button-container" b:select="b:windowhead/div[last()]" />
   <s:render b:test="contains(@b:windowbuttons, 'close')"
             b:destination="$button-container"
             b:mode="asfirstchild">
      <b:window-closebutton />
   </s:render>
   <s:render b:test="contains(@b:windowbuttons, 'maximize')"
             b:destination="$button-container"
             b:mode="asfirstchild">
      <b:window-maximizerestorebutton />
   </s:render>
   <s:render b:test="contains(@b:windowbuttons, 'minimize')"
             b:destination="$button-container"
             b:mode="asfirstchild">
      <b:window-minimizebutton />
   </s:render>
   <s:task b:action="set"
           b:target="$button-container/style::width"
           b:value="{concat(count($button-container/*) * 17, 'px')}" />
</s:if>
```

The first part is an *s:if* tag which contains an XPath statement that tests for the value of the attribute *b:windowbuttons*. When it has the value *none*, no buttons are added, so no action has to be undertaken. When it has any other value, it checks the attribute for specific values.

First, a container for the buttons will be rendered into the *b:windowhead*. This container takes care of the button positioning.

Next, a variable (*$button-container*) that points to the container is defined. This variable is used for defining the destination of the *s:render* tags that render the buttons. Each of the following *s:render* tags is only executed if the *b:windowbuttons* attribute contains a certain value (*close*, *maximize* and *minimize*).

The *s:render* tag is an important tag which can have many uses in dynamic applications. In its simplest form, the contents of the *s:render* tag is simply copied to the target specified by the *b:destination* attribute. However the contents of *s:render* tag can be quite dynamic, through the use of XPath and XSLT-like commands. Refer to the BXML reference for details of which XSLT commands you can use within the *s:render* tag.

When all possible values are checked, and the buttons have been rendered, some styling of the button container occurs. In order to make sure all buttons fit in the container, it counts the number of buttons (represented by *) in the container, multiplies the result with 17 and then adds 'px' to it to create a valid width.

If you examine the rest of the construct behavior, you can see that depending on the custom attributes and their values found on the *b:window* element that is being constructed, different attributes and elements will get set onto the *b:window* element as it is constructed. This makes it possible for a few parameters on the window element to have a large impact on the eventual structure of the window.

The rest of the window controls are composed is a similar way; once you understand how one works, you can more or less understand the others. Once again, this document is in no way meant to provide a complete explanation of how the Windows Starter Kit works, or to cover every aspect of BXML. You are advised to look through the code of Windows Starter Kit yourself and consult the BXML documentation where necessary.