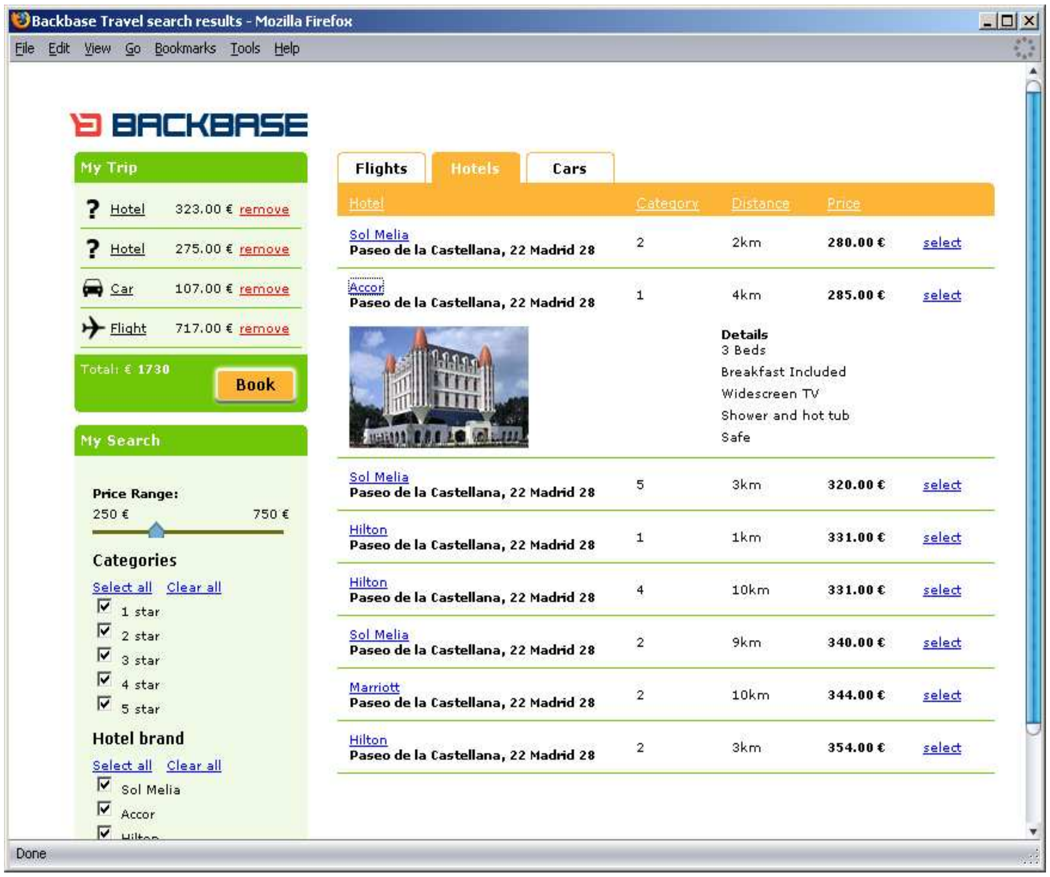
The Travel Starter Kit

1. Introduction

The Travel Starter Kit is an example of how to build a vacation booking application for travel agencies using BXML. The application provides filtering and sorting of flights, hotels and cars. It allows the user to specify his or her preferences, and hides the options that do not match the preferences. Flights, hotels or cars that are selected are added to a shopping cart. The Travel Starter Kit demonstrates how to use custom JavaScript to embellish BXML, and how to visually enhance applications using effects.


Backbase Travel search results - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
BACKBASE
Flights
Hotels
Cars
Book
Done

My Trip

Hotel	323.00 €	remove
Hotel	275.00 €	remove
Car	107.00 €	remove
Flight	717.00 €	remove

Total: € 1730

My Search

Price Range:
250 € 750 €

Categories
Select all Clear all
- ☑ 1 star
- ☑ 2 star
- ☑ 3 star
- ☑ 4 star
- ☑ 5 star

Hotel brand
Select all Clear all
- ☑ Sol Melia
- ☑ Accor
- ☑ Hilton

Hotel	Category	Distance	Price	
Sol Melia Paseo de la Castellana, 22 Madrid 28	2	2km	280.00 €	select
Accor Paseo de la Castellana, 22 Madrid 28	1	4km	285.00 €	select
Sol Melia Paseo de la Castellana, 22 Madrid 28	5	3km	320.00 €	select
Hilton Paseo de la Castellana, 22 Madrid 28	1	1km	331.00 €	select
Hilton Paseo de la Castellana, 22 Madrid 28	4	10km	331.00 €	select
Sol Melia Paseo de la Castellana, 22 Madrid 28	2	9km	340.00 €	select
Marriott Paseo de la Castellana, 22 Madrid 28	2	10km	344.00 €	select
Hilton Paseo de la Castellana, 22 Madrid 28	2	3km	354.00 €	select

Details
3 Beds
Breakfast Included
Widescreen TV
Shower and hot tub
Safe

This document is designed to provide you with an overview of the concepts that are used to implement the starter kit; it is not its intention to describe all the functionality of the application, or to explain in detail how the application is built. It is recommended to consult the Manual or BXML Reference PDF for more detailed information about the BXML code used in the starter kit.

2. The core structure of the Travel Starter Kit

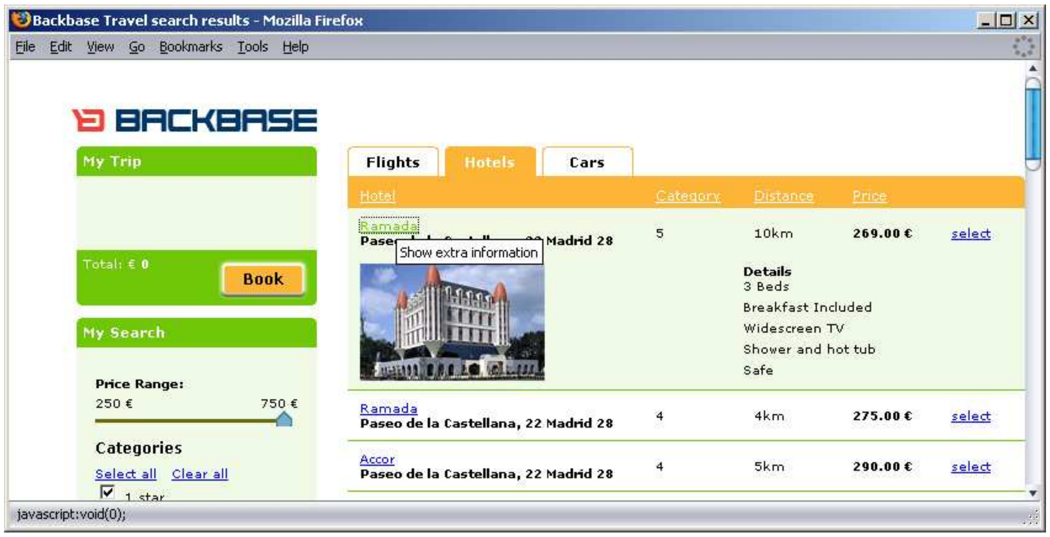
The application can be divided in three functional areas; "My Trip", the Main Application Area (for selecting flights, hotels, and cars), and "My Search".

2.1 My Trip

"My Trip" is located in the upper-left hand corner of the application. This is basically a shopping cart, to which items such as flights, hotels and cars can be added or removed. The shopping cart keeps a running total of the cost of the trip, according to its contents.

2.2 Main Application Area (Tabbed Content)

The main area of the application is tabbed interface with tabs for Flights, Hotels, and Cars. The tabs contain the items that can be selected for the trip. On application startup, only the contents of the initially selected tab are loaded; the contents of the tabs that are not initially displayed are not loaded to reduce initialization time of the application. When a tab is selected for the first time, the contents are dynamically loaded. You will see a loading message when the contents are being loaded for the first time. The message will not appear when a tab is selected again. Similarly, additional information about an item is only added to the application when requested.



Hotel	Category	Distance	Price	
Ramada Pase… Madrid 28	5	10km	269.00 €	select
Details 3 Beds Breakfast Included Widescreen TV Shower and hot tub Safe				
Ramada Paseo de la Castellana, 22 Madrid 28	4	4km	275.00 €	select
Accor Paseo de la Castellana, 22 Madrid 28	4	5km	290.00 €	select

The content of the tabs is generated using PHP; there is one file for each tab. Each PHP file generates an XML file containing BXML. Items that can be selected for the trip are listed in the file. Certain properties, such as the price of the items are generated randomly, so some parts of the contents of the tabs are different every time the application is reloaded.

2.3 My Search

My Search, located under My Trip, allows the user to specify preferences, so the items can be filtered and items displayed according to the preferences. This application offers two ways of changing preferences: by changing a slider value the user can, for example, specify the maximum price he or she wants to spend on a certain part of the trip, or by checking checkboxes that allow the showing or hiding of items that have a particular property value. For example the user can choose to not list hotels with a one star rating. Since filters for flights are not the same as those for hotels, My Search changes when a different tab is selected.

3. Application Architecture

The Travel starter kit uses most of the key BXML concepts, such as include files and behaviors. This chapter will shortly discuss the most important concepts for the starter kit.

3.1 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 regular XHTML. 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 can contain multiple nested include files.

This starter kit uses 5 includes:

```
<s:include b:url="behaviors/checkout.xml" />
<s:include b:url="behaviors/filter.xml" />
<s:include b:url="behaviors/result.xml" />
<s:include b:url="behaviors/tab.xml" />
<s:include b:url="behaviors/sort.xml" />
```

The included files contain behaviors for distinct parts of the application.

3.2 Decks

The deck control is one of the most important concepts in creating single page interfaces. A deck only shows the selected element inside the deck; all other elements are hidden. There are two decks in the Travel starter kit, one containing the items that can be selected for the trip, and the other containing the form elements that take can be used for filtering the items that are displayed in the first deck. Both decks contain `b:buffer` tags that are used to load content on request. The `b:buffer` tag has a `b:url` attribute that points to the file that is loaded when the `b:buffer` is selected.

```
<b:deck id="BBTRAVEL-Search-results">
       <b:buffer b:url="_result/flights.php" id="BBTRAVEL-Flight-results" />
       <b:buffer b:url="_result/hotels.php" id="BBTRAVEL-Hotel-results" />
       <b:buffer b:url="_result/cars.php" id="BBTRAVEL-Car-results" />
</b:deck>
```

3.3 Behaviors

A behavior is a generic construct used to encapsulate functionality, promoting reuse and separating the structure of the document from the behavior. You can use behaviors to:

- Define the visual representation of an element when in one of its base states *selected* or *deselected* (using the `s:state` tag)
- Define the instructions that are executed when an event takes place by using the `s:event` tag to define an event handler

For more information on behaviors, see section ***5 Adding effects***.

4. Using JavaScript to enhance BXML

BXML offers a rich library of functionality using XPath and the DOM. It is also completely interoperable with web standards, which means you can still use JavaScript to add custom functionality, and make use of existing JavaScript functions in BXML applications. This section describes a situation where using JavaScript provides an appropriate solution.

The *Outgoing arrival time* slider, in the Flights section of the application, is one element in the application that uses JavaScript. The slider can be used to filter the items shown in the Flights tab, and while sliding, a tooltip shows the currently selected value.

Let's look at the slider code (*_filter/_filter_flights.xml – line 36*):

```
<b:slider
        b:behavior="filter-slider"
        w:filtertype="upper"
        w:prop="outgoing_arrival"
        w:section="Flights"
        w:postfix="H"
        b:value="2400"
        b:gfxset="WHP"
        b:ori="hor"
        b:start="0"
        b:end="2400"
        b:snap="true"
        b:step="100">

        <s:event b:on="showtooltip">
                <s:task
                        b:action="set"
                        b:target="id('slidervalue')/text()"
                        b:value="{getTooltipValue(string(@b:value))}" />
                <s:task b:action="show" b:target="id('slidervalue')" />
                <s:task
                        b:action="position"
                        b:target="id('slidervalue')"
                        b:type="place"
                        b:position="at-pointer" />
        </s:event>
</b:slider>
```

This slider has a behavior (`filter-slider`*)*, a default value of `2400`, a minimum value of `0`*,* and a maximum value of `2400`. Steps are defined at `100`, meaning that the slider is divided in 24 values, representing the hours of a day. Then the `showtooltip` event is defined. This event overrides the *showtooltip* event that is defined in the `filter-slider` behavior. This is done because the format of the tooltip content is different from the default behavior. The default behavior would be to just to print the current value of the slider, which is the value of the `b:value` attribute. This attribute contains numbers (0 - 2400). The tooltip, on the other hand, shows a formatted time; *00:00 – 24:00*. A transformation therefore has to be made to show the right format.

The right format is a four digit number, so values less than 1000 need to be adapted. This comes down to adding a *0* in front of the value when there are three digits to account for values between 100 and 900. Since the value can only be a multiple of 100, starting at 0, the only exception is 0. Since it is 0, it can just be substituted by 0000.

When the amount of digits is correct, a colon is added for final formatting purposes. You *can* do this using BXML and XPath only. Focusing purely on the formatting and setting of the tooltip content, it could be implemented as follows:

```
<s:event b:on="example-showtooltip">
        <!-- create variable, because the attribute value should not be changed -->
        <s:variable b:name="time" b:select="@b:value" />
        <!-- if there is only one digit, set to '0000' -->
        <s:task
           b:test="string-length(string($time)) = 1"
           b:action="assign"
           b:target="$time"
           b:select="'0000'" />
        <!-- if there are three digits, add one more digits -->
        <s:task
           b:test="string-length(string($time)) = 3"
           b:action="assign"
           b:target="$time"
           b:select="concat('0',$time)" />
        <!-- Add the colon -->
        <s:task
           b:action="assign"
           b:target="$time"
           b:select="concat(substring($time, 1, 2), ':', substring($time, 3))" />
        <!-- set tooltip value -->
        <s:task b:action="set" b:target="id('slidervalue')/text()" b:value="{$time}"
/>
        <!-- etc... -->
</s:event>
```

Although the code works perfectly well, and is correct and fast, you can also use JavaScript to reach the same objective. As you can see, the `showtooltip` event that is used in the slider is much smaller. In fact, the five lines in the example do the same as the first line of BXML code in the `settooltip` event:

```
<s:task
        b:action="set"
        b:target="id('slidervalue')/text()"
        b:value="{getTooltipValue(string(@b:value))}" />
```

This is because JavaScript does the formatting. The *set* command is used to set the text of the slider. The value that is used to set the text is specified in the `b:value` attribute. The `{getTooltipValue(string(@b:value))}` query consists of the following:

- `@b:value` - a BXML attribute
- `string()` – an XPath function
- `getTooltipValue()` - a JavaScript function
- `{}` - curly brackets

`getTooltipValue()` is defined in the file `js/general.js`:

```
function getTooltipValue( aValue ) {
        var sValue = aValue[0];
        if(sValue.length == 1) sValue = '0000';
        else if(sValue.length == 3) sValue = '0' + sValue;
        return sValue.substr(0,2) + ':' + sValue.substr(2);
}
```

The first line of `getTooltipValue()` takes the first value of an array. This means that the input of the function is expected to be an array. The XPath query returns a string, so why expect an array? It is important to note that XPath queries always return a sequence. A sequence can contain multiple strings, integers, Booleans, and objects (nodes). In BXML, this sequence is stored in an array. So even when there is only one result, the result is always be in an array.

The argument that is given to the `getTooltipValue()` is the result of the XPath query `string(@b:value)`. The result of that query is the string value of the slider. The string function is necessary, since it passes the object (the attribute itself) if a `@b:value` is specified.

The curly brackets indicate that the value requires parsing. If the curly brackets were omitted, the contents would not be evaluated as an XPath query, but would instead be interpreted as a string.

Although the example is a basic one - there is only a small change in formatting and the logic is kept simple – it serves as an example that you can extrapolate to create more complicated functions, and demonstrates how you can use JavaScript to do similar transformations or advanced calculations that can help you keep your code clean and compact.

5. Adding effects

Adding effects to a web page gives the application a more interactive feel than a visually static page. You can add effects to make an application look cool, or just to add a bit of style. Effects are often relatively small and unobtrusive, yet nevertheless add significantly to the usability of the application. Not having them would reduce the overall user-experience of the application.

You can create several kinds of visual effects using BXML: elements can be moved or resized to change the layout of a page, background, text or border colors can be changed to draw attention, and opacity can be changed to fade in or fade out elements. There are several elements in the starter kit with visual enhancements, one of which, the *Book* button, will be discussed in this section.

The *Book* button uses a mouseover effect (similar to an effect used in the tabs) that:

- Indicate when the button is pressed
- Fade color fade when the mouse enter/leave the button

The button's behavior that contains the code that creates these effects is located in the *behaviors/checkout.xml*:

```
<s:behavior b:name="BBT-Checkout-Button">
        <s:state
                b:on="deselect"
                b:normal="BBT-Checkout-Button"
                b:press="BBT-Checkout-Button BBT-Checkout-Button-press" />
        <s:initatt b:textselect="false" />
        <s:event b:on="mouseenter">
                <s:fxstyle
                        b:color="#FCB537"
                        b:background-color="#FFFFFF"
                        b:time="200" />
        </s:event>
        <s:event b:on="mouseleave">
                <s:fxstyle
                        b:color="#000000"
                        b:background-color="#FCB537"
                        b:time="200" />
        </s:event>
</s:behavior>
```

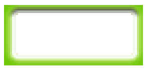
There is one `s:state` tag, two `s:event` tags, and an `s:initatt` tag (the later is not part of either effect — it is used merely to add initial attributes to the element that uses the behaviour — and so will not be discussed.

5.1 Using s:state

The `s:state` tag is used to specify which style classes an element uses, depending on its state. An element can be in one of the two basic states `select` or `deselect`. The `s:state` tag has a `b:on` attribute to specify for which state you are defining which classes to use. Because there is no need to change the state of the button, only the `deselect` (the default state of an element) needs to be defined. The `b:normal` attribute contains the classes that are assigned to the element in its default state. The `b:press` attribute contains the classes that are used when the element is pressed (a mouse button is pressed down over the element).

You can create effects using this concept of changing an element's classes based on its state. Take, for example, the button used in the application. A background image has been attached to the element to make it look like a button. The image has a 3D look that is created by applying shadows and highlights. When you press a button, the shadows and highlights change. The button looks like it's changing the highlights, but in fact, a different image with different highlights is attached to the button, depending on the state of the mouse (pressed or not). The following image is used for the normal state of the button, as defined in `BBT-Checkout-Button`. The shadow is on the bottom side.

The following image is used for the pressed state of the button, as defined in `BBT-Checkout-Button-press`. The shadow is on the top side.



Furthermore, as well as the change of background image, padding is also applied to move the text a few pixels to make it look like the text is printed on the surface of the button. As the surface seems to move, the text on the surface also moves.

Changing style properties by setting different classes is a good way to create certain effects. The different style definitions are applied immediately when a class has been changed, removed, or added, making an application highly-responsive leading to a richer user experience.

5.2 Animated effects – s:fxstyle

Although in many situations changing classes using `s:state` is perfectly adequate, sometimes a more subtle approach is required using a transition effects of one visual state to another.

The *Book* button and the tabs both use a background color fade effect when the mouse is hovered over them. For obvious reasons, the `mouseenter` and `mouseleave` events are used for executing the effects. The effects themselves are created using the `s:fxstyle` tag. The `s:fxstyle` tag has attributes that correspond with CSS style properties that you can set. In the Book button example, `b:color` and `b:background-color` are used. The button has an orange (`#FCB537`) surface, which is specified in the CSS class of the button. When the mouse enters the button, the `mouseenter` event is executed, which contains the following line:

```
<s:fxstyle b:color="#FCB537" b:background-color="#FFFFFF" b:time="200" />
```

This would have the same end result as using `s:setstyle` with the same attributes. However, using this tag, the changes are made gradually. Initially, the button text color is black. The b:color attribute is set to a hexadecimal value that corresponds with the color orange. Thus, when a mouse enters the button, the text color will gradually change to orange. A similar transition is applied to the background color. Where the background color initially was orange, the new color will be white (`#FFFFFF`). The `b:time` attribute specifies how many milliseconds the transition should last. Thus, when a mouse enters the button, it will take 200 milliseconds for the color and background color to be changed to orange and white respectively. This would happen in parallel, since all attributes on the same `s:fxstyle` tag are changed at the same time. If a situation asks for one transition to be finished before the next starts, a second `s:fxstyle` tag can be used. Of course, the `mouseleave` event will do the transition the other way around, so the button will return to its original state.

Note: For more information about the many properties that can be animated using `s:fxstyle`, see the BXML Reference PDF.

6 Conclusion

This starter kit focussed on demonstrating two features of BXML that can make a significant difference to your web applications: you can use existing JavaScript functionality alongside BXML to create advanced applications (note also you can use the `s:script` tag to add inline JavaScript), and you can enhance your application with effects using the `s:state` and `s:fxstyle` tags.

This document is designed to give you an idea about what can be achieved with JavaScript and visual effects by showing how the starter kit uses this functionality, but is not a comprehensive guide to these subjects. For more information, refer to the Manual, Reference, and other Starter Kit documentation. If you have any questions, you can post them on the Backbase forum: http://www.backbase.com/forum