KPR Templates

1. Introduction

The KPR Templates extension patches most KPR constructors to provide dictionary based constructors and templates.

The objective is to simplify the coding of KPR applications and shells in ECMAScript only, thanks to a mechanism similar to what is already available in XML.

1. Overview
   1. Dictionary Based Constructors

KPR constructors take several optional arguments to initialize the properties of the new object they create. That obviously requires remembering the order of the arguments. Moreover, KPR constructors do not initialize all properties of the new object, so some properties have to be initialized separately.

var redSkin = new Skin("red");  
var redContent = new Content({ width:40, height:40 }, redSkin);   
redContent.active = true;   
redContent.visible = true;

A dictionary based constructor takes one argument, which is an object with properties. The constructor uses such properties to initialize the properties of the new object it creates.

var greenSkin = new Skin({ fill: "green" });  
var greenContent = new Content({  
 width:40, height:40, skin:greenSkin, active:true, visible:true   
});

A dictionary can also describe a containment hierarchy or a rich text. For instance:

var scroller = new Scroller({   
 left:0, width:160, top:0, height:120,   
 clip:true, active:true,  
 contents: [  
 new Column({   
 left:0, right: 0, top:0,  
 contents: [  
 new Label({ left:0, right:0, string: "one" }),  
 new Label({ left:0, right:0, string: "two" }),  
 new Label({ left:0, right:0, string: "three" })  
 ]  
 })  
 ]  
});

var text = new Text({  
 left:0, width:160, top:0,  
 blocks: [  
 { string: "block" },  
 { spans: [  
 { wrap: new Picture({   
 url: "<http://www.kinoma.com/img/kinoma-logo.png>"  
 })},  
 { string: "span" },  
 { behavior: linkBehavior, string: "link",  
 style: new Style({ color: "blue" })}  
 ]}  
 ]  
});

For the sake of compatibility, the original KPR constructors are used when there are no arguments, when there are several arguments, or when the single argument is no object.

* 1. Templates

In XML, KPR provides templates; see KPR Overview, 4.3 Templates. In ECMAScript, templates are just constructors. They can be coded explicitly but it is cumbersome, especially when templates are based on templates.

Contents constructors provide the template function to create templates. The template function takes one argument, which is an anonymous function that creates a dictionary. The template function returns a constructor.

var MyLabel = Label.template(function($) { return {  
 left:0, right:0, string:$   
}});

When calling such a constructor with new and data, the anonymous function is called with the data to create the dictionaries necessary to instantiate the contents.

var oopsLabel = new MyLabel("oops");  
var wowLabel = new MyLabel("wow");

Templates can be chained.

var MyGreenLabel = MyLabel.template(function($) { return {  
 skin:greenSkin   
}});

In that case, the anonymous functions are called from the most generic to the most specific. Properties of the dictionaries are overridden in that order, except for the contents properties, which are concatenated in that order.

Notice that templates do not build a prototype hierarchy. The prototype property of a template always equals the prototype property of the constructor it is based on.

// MyGreenLabel.prototype == Label.prototype

* 1. Instructions

In XML, KPR provides the iterate, scope and select instruction elements. See KPR Overview, 4.3.3 Instructions. Various ECMAScript expressions can be used instead. For instance:

iterate

XML

<scroller id="MyScroller" left="0" width="160" top="0" height="120">  
 <column left="0" right="0" top="0">  
 <iterate on="$">  
 <label like="MyLabel"/>  
 </iterate>  
 </column>  
</scroller>

ECMAScript

var MyScroller = Scroller(function($) { return {  
 left:0, width:160, top:0, height:120,   
 contents: [  
 new Column({  
 left:0, right: 0, top:0,  
 contents: $.map(function($$) {  
 return new MyLabel($$);  
 })  
 })  
 ]  
}});

application.add(new MyScroller(["one","two","three","four"]));

scope

XML

<container id="MyHeader" left="0" width="160" top="0" height="40">  
 <scope with="$.title">  
 <label like="MyLabel"/>  
 </scope>  
</container>

ECMAScript

var MyHeader = Container(function($) { return {   
 left:0, right:160, top:0, height:40,  
 contents: [  
 new MyLabel($.title);  
 ]  
}});

application.add(new MyHeader({ title: "five" }));

select

XML

<container id="MyHeader" left="0" width="160" top="0" height="40">  
 <select on="$.title">  
 <label like="MyLabel" with="$.title"/>  
 </select>  
</container>

ECMAScript

var MyHeader = Container(function($) { return {  
 left:0, right:160, top:0, height:40,  
 contents: $.title ? [ new MyLabel($.title) ] : undefined  
}});

* 1. Constructors Called as Functions

Dictionary based constructors take one argument: the dictionary.

var myContainer = new Container({ width:160, height:40 });

Template based constructors take one argument: the data.

var myHeader = new MyHeader({ title: "wow" })

When both kinds of constructors are called as functions (without new), they take two arguments: the data and the dictionary, and return an instance of their prototype property.

Calling a template based constructor as a function is useful to override properties.

var myHeader = MyHeader({ title: "wow" }, { height:50 })

Calling a dictionary based constructor as a function is useful to pass data to its behavior.

var myContainer = Container({ title: "wow" }, {  
 behavior: {  
 onCreate: function(container, $) {  
 // $.title == "wow"  
 }  
 }  
})

And, mostly, it helps to define templates that construct containment hierarchies without repetitive new and with consistent arguments for all constructors.

var MyScreen = Container.template(function($) { return {  
 left:0, right:0, top:0, bottom:0,  
 contents: [  
 Container($, { anchor:"BODY",  
 left:0, right:0, top:44, bottom:0,  
 behavior: {  
 onCreate: function(container, data) {  
 this.data = data;  
 },  
 },  
 contents: [  
 SCROLLER.VerticalScroller($, { clip:true,  
 contents:[  
 MyColumn($),  
 SCROLLER.TopScrollerShadow($),  
 SCROLLER.BottomScrollerShadow($),  
 ]}),  
 ]  
 }),  
 MyHeader($, { height:44 }),  
 ],  
}});

Notice also that the onCreate event is no side effect of the Behavior constructor but is triggered when the content object has been created.

* 1. Shortcut

Handlers are binding a path to a behavior in order to receive messages, see the KPR Overview, 7.2 Handlers. The bind function of the Handler constructor is an explicit shortcut:

Handler.bind("/wow", {  
 onInvoke: function(handler, message) {  
 debugger  
 }  
}));

is equivalent to:

var wowHandler = new Handler("/wow");  
wowHandler.behavior = {  
 onInvoke: function(handler, message) {  
 debugger;  
 }  
}  
Handler.add(wowHandler);

1. Reference
   1. Constructors

The Canvas, Column, Content, Container, Label, Layer, Layout, Line, Media, Picture, Port, Scroller, Skin, Style,Text and Thumbnail constructors are dictionary based constructors.

Constructor(dictionary)

dictionary object

An object with properties to initialize the result

Returns object

An instance of *Constructor*.prototype

* 1. Templates

The Canvas, Column, Content, Container, Label, Layer, Layout, Line, Media, Picture, Port, Scroller, Text and Thumbnail constructors provide the template function.

Constructor.template(anonymous)

anonymous function

A function that returns an object with properties to initialize the instances that the result creates

Returns function

A constructor that creates instances of *Constructor*.prototype

The prototype property of the result equals *Constructor*.prototype.

The result also provides the template function.

* 1. Dictionaries

Here are the properties that the dictionaries can contain. All properties are optional. For details about the properties, see KPR ECMAScript API Reference.

Canvas Dictionary

Same as the content dictionary

Column Dictionary

Same as the container dictionary

Content Dictionary

active boolean

If true, the content can be touched

backgroundTouch boolean

If true, the content can be touched in the background

behavior object

The content's behavior

bottom number

The content's bottom coordinates

duration number

The content's duration in milliseconds

fraction number

The content's fraction,

exclusiveTouch boolean

If true, the content captures the touch

height number

The content's height coordinates

interval number

The resolution of the content's clock in milliseconds

left number

The content's left coordinates

name string

The content's name

right number

The content's right coordinates

skin object

The content's skin, as an instance of Skin.prototype

state number

The content's state

style object

The content's style, as an instance of Style.prototype

time number

The content's time in milliseconds

top number

The content's top coordinates

variant number

The content's variant

width number

The content's width coordinates

Container Dictionary

Same as the content dictionary, plus:

clip boolean

If true, the container crops its contents

contents array

An array of contents

Label Dictionary

Same as the content dictionary, plus:

editable boolean

If true, the label's string can be edited by users

hidden boolean

If true, the label's string is hidden to users

selectable boolean

If true, the label's string can be selected by users

string string

The label's string

Layout Dictionary

Same as the container dictionary

Layer Dictionary

Same as the container dictionary, plus:

alpha boolean

If true (the default) the layer needs an alpha channel

effect object

The layer's effect, as an instance of Effect.prototype

Line Dictionary

Same as the container dictionary

Media Dictionary

Same as the content dictionary, plus:

aspect string

The media's aspect as draw, fill, fit or stretch

mime string

The media's MIME type

url string

The media's URL

Picture Dictionary

Same as the content dictionary, plus:

aspect string

The picture's aspect as draw, fill, fit or stretch

effect object

The picture's effect, as an instance of Effect.prototype

mime string

The picture's MIME type

url string

The picture's URL

Port Dictionary

Same as the content dictionary

Scroller Dictionary

Same as the container dictionary, plus:

loop boolean

If the scroller's is looping

Skin Dictionary

For color skins:

borders object

The skin's borders, an object with left, right, top or bottom number properties

fill string/array

The color to fill content object with, as one string or an array of strings that define colors as in CSS

stroke string/array

The color to stroke content object with, as one string or an array of strings that define colors as in CSS

For texture skins:

aspect string

The skin's aspect as draw, fill, fit or stretch

margins object

The skin's margins, an object with left, right, top or bottom number properties

states number

The vertical offset between variants

texture string

The skin's texture, a texture object

tiles object

An object with left, right, top or bottom number properties to make a 1-part, 3-part, or 9-part patterns

variants number

The horizontal offset between variants

x number

y number

width number

height number

The portion of the texture object to extract

Style Dictionary

bottom number

The style's bottom margin

color string/array

The style's color, as one string or an array of strings that define colors as in CSS

font string

The style's font, as a string that defines a font in CSS

horizontal string

The style's horizontal alignment, as left, center, right or justify

indentation number

The style's indentation: the indentation of the first line of a block

leading number

The style's line height: distance between lines of a block

left number

The style's left margin

lines number

The style's line count: maximum number of lines in a block

right number

The style's right margin

size number

The style's size

top number

The style's top margin

vertical string

The style's vertical alignment, as top, middle or bottom

Text Dictionary

Same as the content dictionary, plus:

blocks array

An array of blocks. A block is an object with behavior, style and string properties: - - The behavior property is an object or null (the default). When the text is active and the block is touched, it calls the corresponding function properties of its behavior.

- The style property is an instance of Style.prototype or null (the default).

- The string property is a string.

Instead of a string property, a block can have a spans property. The spans property is an array of spans or wraps. Like a block, a span is an object with behavior, style and string properties. A wrap is an object with content and align properties:

- The required content property is any content object.

- The align property is a string as left, right, top, middle (the default), or bottom.

editable boolean

If true, the text's string can be edited by users

hidden boolean

If true, the text's string is hidden to users

selectable boolean

If true, the text's string can be selected by users

string string

The text's string

Thumbnail Dictionary

Same as the picture dictionary

* 1. Shortcut

Handler.bind(path, behavior)

path object

The path of the handler object

behavior object

The handler's behavior

Creates a handler object with path; assigns its behavior with behavior; puts the handler in into the set of active handler objects.