

HAXEUI VERSION 2

MULTI-PLATFORM, MULTI-FRAMEWORK UI



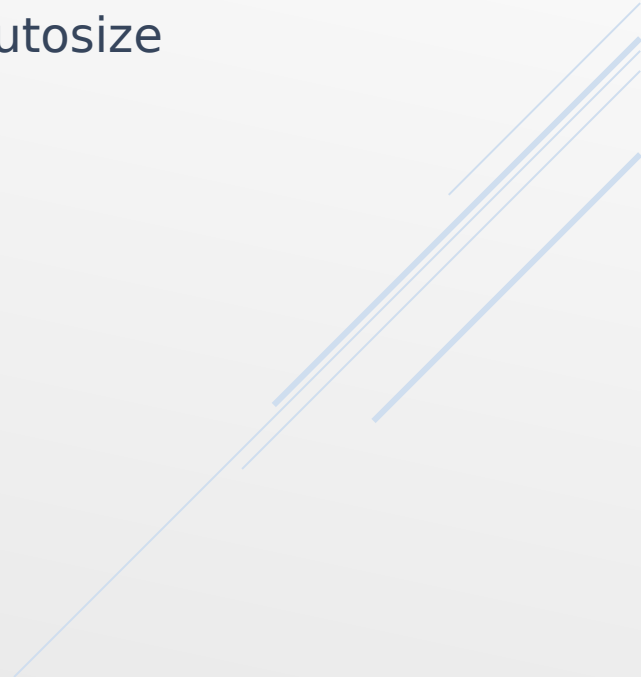
The Good

- Easy to setup and use
- Decent component set
- XML UIs
- Macro based controllers
- Scriptable
- Style system

The Bad

- Integration with existing apps
- Little to no documentation
- No HTML5 support
- Not render agnostic
- No binding support
- Text input

The Ugly

- Themes
 - Class hierarchy
 - CSS format
 - Autosize
- 
- Several thin, parallel blue lines of varying lengths and orientations are positioned in the bottom right corner of the slide, creating a modern, abstract design element.

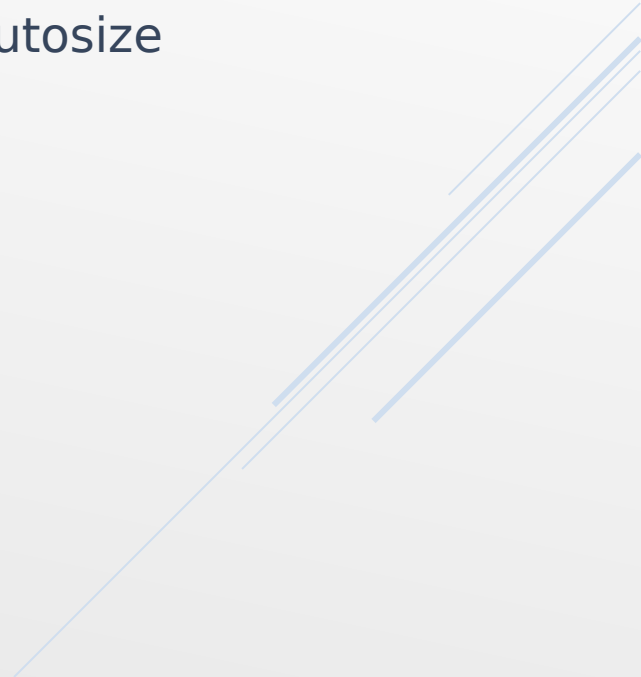
The Good

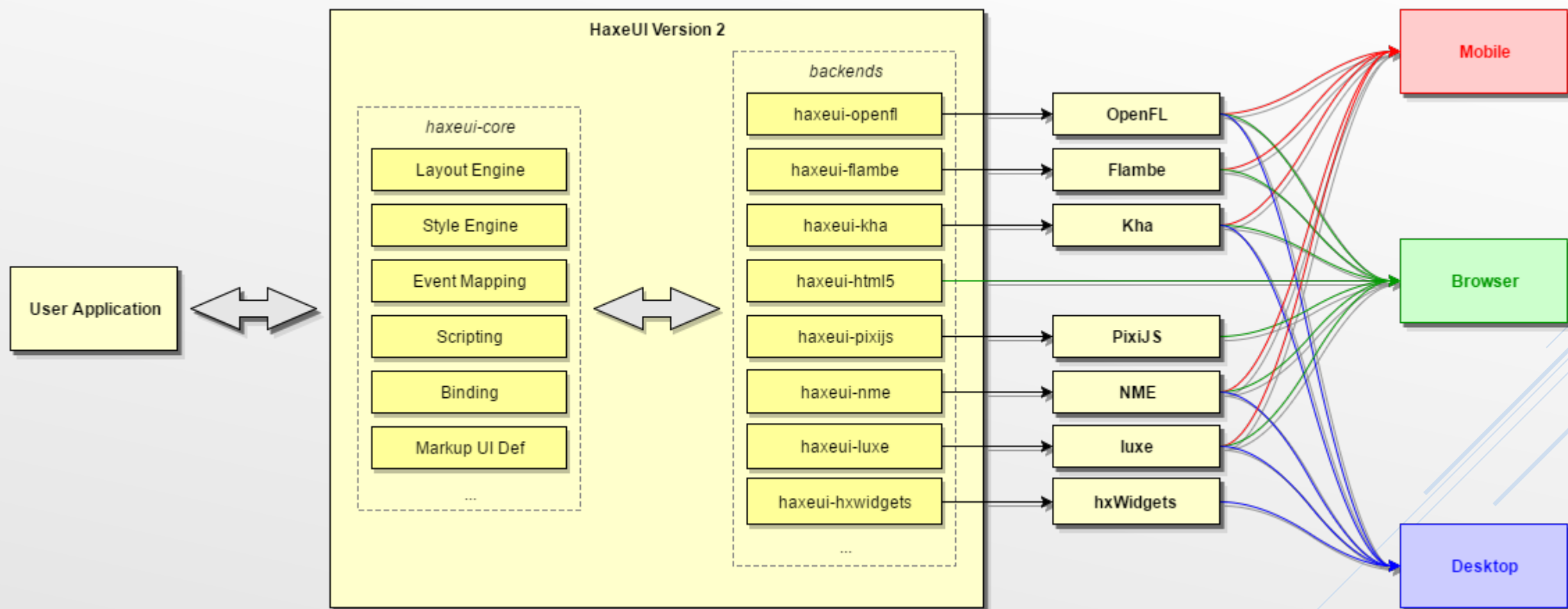
- Easy to setup and use
- Decent component set
- XML UIs
- Macro based controllers
- Scriptable
- Style system

The Bad

- Integration with existing apps
- Little to no documentation
- No HTML5 support
- Not render agnostic
- No binding support
- Text input

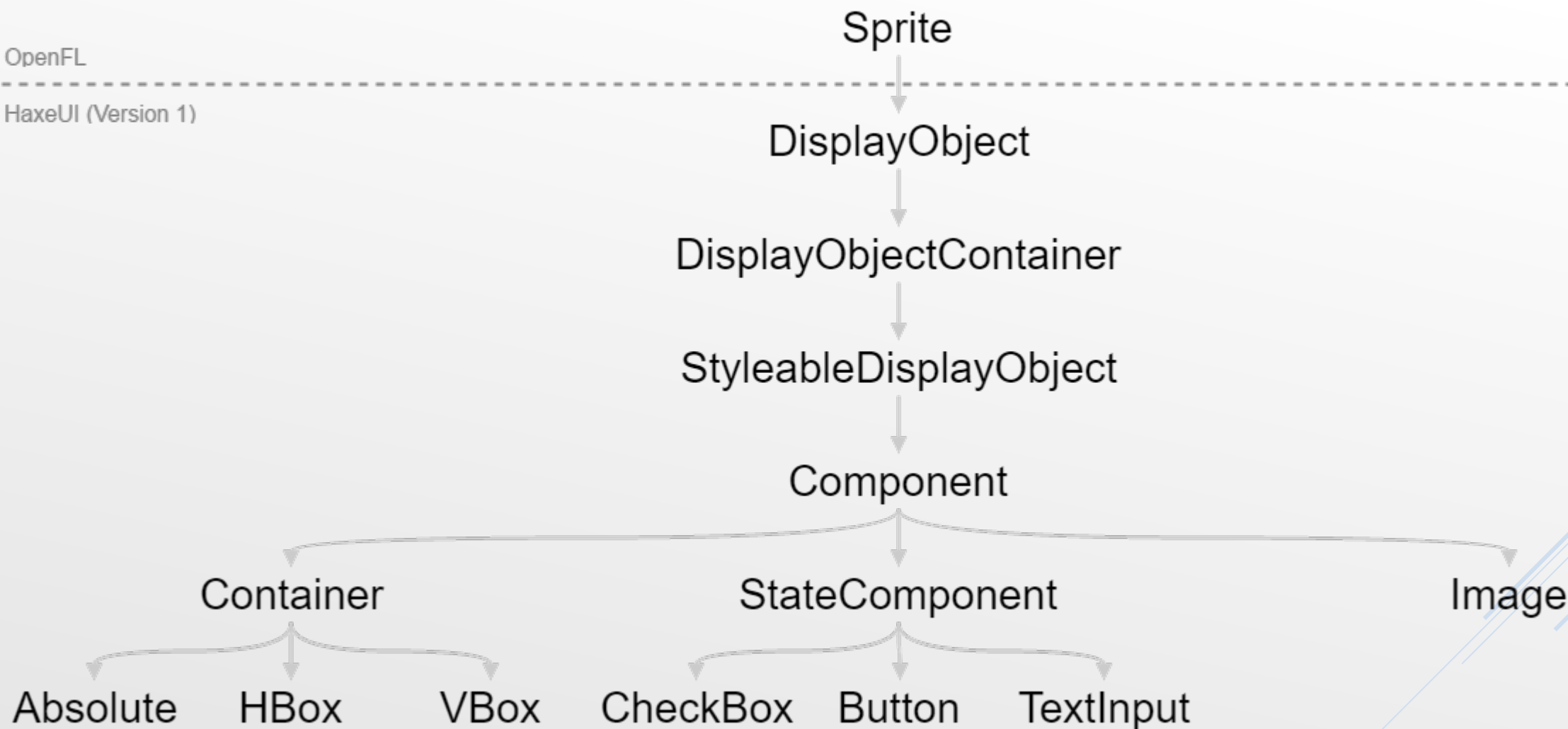
The Ugly

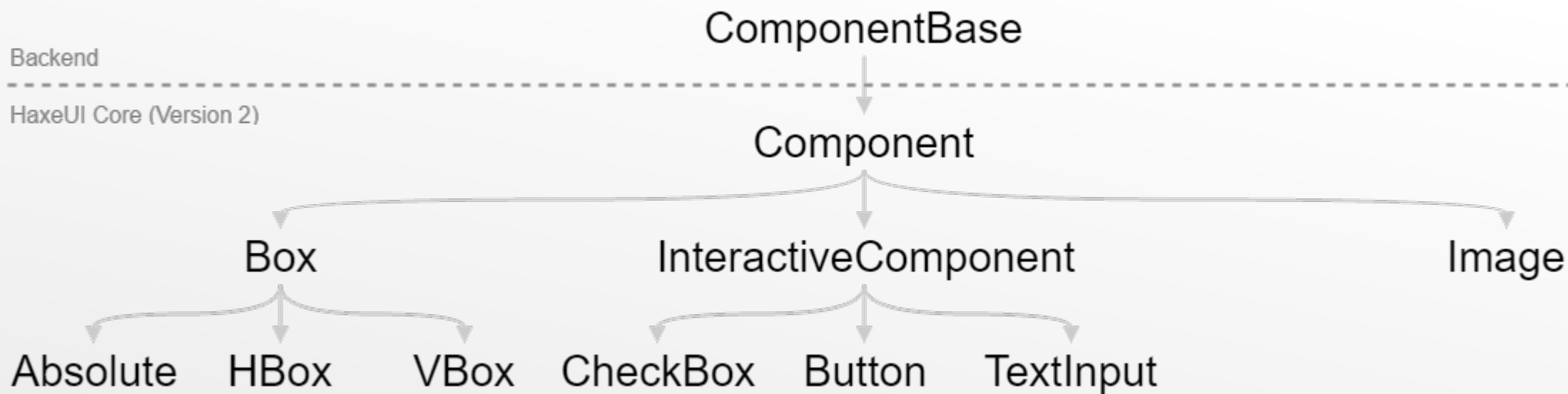
- Themes
 - Class hierarchy
 - CSS format
 - Autosize
- 
- Several thin, parallel blue lines of varying lengths and orientations are positioned in the bottom right corner of the slide, creating a decorative graphic element.



OpenFL

HaxeUI (Version 1)





Backends

- **Auto discovered (.config.xml)**
- **Class replacement**

flambe.config.xml

```
<classes>
  <class source="flambe.display.Texture"      target="haxe.ui.assets.ImageData" />
  <class source="haxe.ui.flambe.ComponentBase" target="haxe.ui.core.ComponentBase" />
  <class source="haxe.ui.flambe.ScreenBase"    target="haxe.ui.core.ScreenBase" />
  ...
</classes>
```

- **Native config**
 - Behaviour substitution
 - Size reporting
 - Custom layouts

html5.config.xml

```
<component id="haxe.ui.components.Button" class="haxe.ui.html5.native.NativeElement" nodeType="button" style="padding:0px; padding-bottom: 1px">
  <behaviour id="text" class="haxe.ui.html5.native.behaviours.SpanText" style="margin-top:-2px;margin-left:-2px;" />
  <behaviour id="icon" class="haxe.ui.html5.native.behaviours.ElementImage" />
  <layout class="haxe.ui.html5.native.layouts.ButtonLayout" />
</component>
```

hxwidgets.config.xml

```
<component id="haxe.ui.components.Button" class="hx.widgets.Button">
  <behaviour id="text" class="haxe.ui.hxwidgets.behaviours.ControlLabel" />
  <behaviour id="icon" class="haxe.ui.hxwidgets.behaviours.ControlBitmap" />
  <size class="haxe.ui.hxwidgets.size.BestSize" includePadding="false" />
</component>
```

- **Resources**

- Package haxe resources
- Recursive discovery
- Allow prefixing

module.xml

```
<resource path="/haxe/ui/_module/styles" prefix="haxeui-core/styles" />
```

usage in .css file

```
.hscroll .button.deinc {  
    icon: "haxeui-core/styles/default/left_arrow.png";  
}
```

usage UI def

```
<image resource="haxeui-core/styles/default/left_arrow.png" />
```

- **Components**

- Define component classes
- Define component packages
- Allow component aliasing

module.xml

```
<components>  
    <class name="haxe.ui.core.Component" />  
    <class package="haxe.ui.components" />  
    <class package="haxe.ui.containers" />  
    <class name="haxe.ui.components.Label" alias="text" />  
</components>
```


- **Scriptlets**

- Allow short form class construction
- Force “keep” on classes & packages
- Automatically add static references

module.xml

```
<scriptlets>
  <import class="haxe.ui.core.Component" keep="true" />
  <import package="haxe.ui.components" keep="true" />
  <import class="Std" keep="true" static="true" />
  <import class="Math" keep="true" static="true" />
</scriptlets>
```

usage in hscript

```
var button = new Button();
button.text = "Button " + Std.string(Math.random());
```

- **Themes**

- Define themes & resources
- Add to existing themes
- Extend themes

module.xml

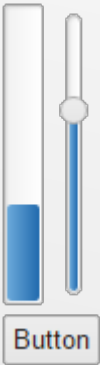
```
<themes>
  <global>
    <style resource="haxeui-core/styles/global.css" />
  </global>
  <default>
    <style resource="haxeui-core/styles/default/main.css" />
  </default>
  <myTheme parent="default">
    <style resource="myTheme.css" />
  </myTheme>
</themes>
```

Resources

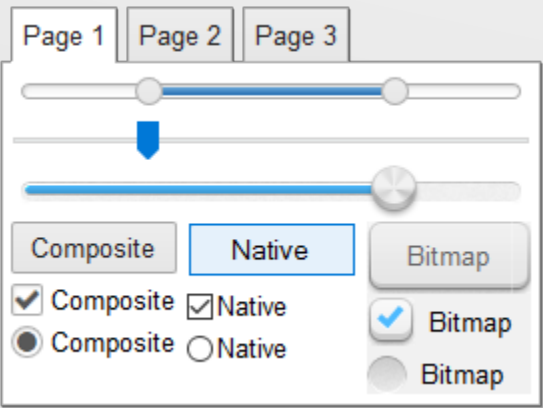
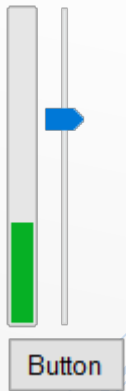
- **Always asynchronous**
 - `ToolkitAssets.getImage(resourceId:String, callback:ImageInfo->Void, useCache:Bool = true)`
- **Two forms**
 - `haxe.Resource` (*haxe.io.Bytes*)
 - Framework (backend) native
- **Converted to framework specific class**

Framework	Resource Class
Flambe	<i>flambe.display.Texture</i>
HTML5	<i>js.html.ImageElement</i>
hxWidgets	<i>haxe.Resource</i>
Kha	<i>kha.Image</i>
lux	<i>phoenix.Texture</i>
NME	<i>nme.display.BitmapData</i>
OpenFL	<i>openfl.display.BitmapData</i>
PixiJS	<i>pixi.core.textures.Texture</i>

- **Type of resource is transparent**
 - Framework resources take precedence



Composite Components	Native Components
Built entirely out of HaxeUI components	Creation delegated to backend
Themes are applicable	Themes most likely selective at best
Consistent yet custom look & feel	Look & feel matches host operating system
Every visible aspect configurable via CSS	Hard to create totally custom UIs
Logic and layout handled via core	Reports size back into core (if auto-sized)



• Hybrid User Interfaces

- Mix and match composite and native components
- Get the best of both worlds
- “Fill in gaps” in native toolkits

```
// existing application
var bmpData:BitmapData = Assets.getBitmapData("img/haxe_logos.png");
var bmp:Bitmap = new Bitmap(bmpData);
Lib.current.stage.addChild(bmp);

Toolkit.openPopup({x:10, y:10, width: 100, height:100}, function(root:Root) {
    var button:Button = new Button();
    button.text = "Button";
    button.x = 10;
    button.y = 10;
    root.addChild(button);
});
```



- Floating over the top (can be changed with styles)
- Client app must handle resizing of root
- Hard to add “just a button” to existing application

- OpenFL – extends **Sprite**

```
var button:Button = new Button();
button.text = "Button";
Lib.current.stage.addChild(button);
```

- HTML5 – contains **HTMLElement**

```
var button:Button = new Button();
button.text = "Button";
Browser.document.appendChild(button.element);
```

- Kha – exposes **renderTo** function

```
var g = framebuffer.g2;
var button:Button = new Button();
button.text = "Button";
button.renderTo(g);
```

- Flambe – extends **Sprite**

```
var button:Button = new Button();
button.text = "Button";
System.root.addChild(new Entity().add(button));
```

- Luxe – contains **geometry**

```
var button:Button = new Button();
button.text = "Button";
```

- Pixijs – extends **Graphics**

```
var stage = new Container();
var button:Button = new Button();
button.text = "Button";
stage.addChild(button);
```

- Universal #1 – Presupposes everything is **setup in init**

```
var button:Button = new Button();
button.text = "Button";
Screen.instance.addComponent(button);
```

- Universal #2 – Handles the **entire lifecycle**

```
var app = new HaxeUIApp();
app.ready(function() {
    var button:Button = new Button();
    button.text = "Button";
    app.addComponent(c);
});
```

- **Removes a lot of unnecessary boilerplate code**
- **Takes the form of “*source*” and “*target*”**
- **Can bind to multiple “*targets*”**
- **Can specify component property** (defaults to special “*value*” property)
- **Allows for transformation of values** (eg: “*Value is: $\${value * 2}$* ”)
- **Two forms**
 - *Inline*

```
<hslider id="hslider1" pos="50" />
<hslider bindTo="hslider1" bindTransform="${value / 4 * 3}" />
```

- *Normal*

```
<bind source="button-width" target="test-button.width" />
<bind source="button-height" target="test-button.height" />

<hslider id="button-width" min="100" max="200" pos="100" />
<hslider id="button-height" min="100" max="200" pos="100" />

<button id="test-button" icon="img/slinky_tiny.jpg" text="Test Button" />
```

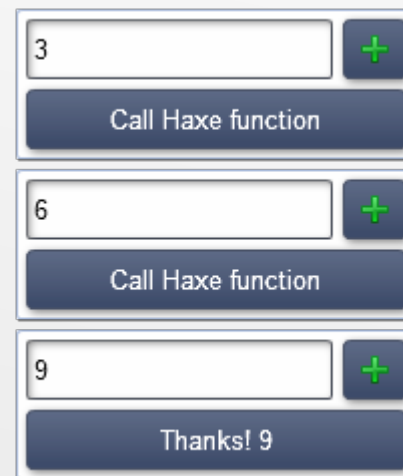
```
<vbox width="200" style="padding: 5px; borderSize: 1px;">
  <hbox width="100%">
    <textinput id="counter-field" width="100%" text="0" />
    <button icon="icons/plus.png" height="100%" />
  </hbox>

  <button id="haxe-button" text="Call Haxe function" width="100%" />
</vbox>
```

```
@:build(haxe.ui.toolkit.core.Macros.buildController("assets/ui/counter.xml"))
class CounterController extends XMLController {
  public function new() {
    add.onClick = function(e) {
      counterField.text = '${Std.parseInt(counterField.text) + 1}';
    }

    haxeButton.onClick = function(e) {
      haxeButton.text = "Thanks! " + counterField.text;
    }
  }
}
```

```
Toolkit.open(function(root:Root) {
  root.addChild(new CounterController().view);
  root.addChild(new CounterController().view);
  root.addChild(new CounterController().view);
});
```



```
<vbox width="200" id="custom" style="padding: 5px; border:1px solid #CCCCCC;">
  <script>
    var counter = Std.parseInt(counterField.text);
    function add() {
      counter++;
      counterField.text = counter;
    }
  </script>

  <hbox width="100%">
    <textfield id="counter-field" width="100%" text="0" />
    <button icon="icons/plus.png" onclick="add()" />
  </hbox>

  <button id="haxe-button" text="Call Haxe function" width="100%" />
</vbox>
```

```
@:build(haxe.ui.macros.ComponentMacros.build("assets/ui/demo/custom/counter.xml", "counter"))
class Counter extends Component {
  public function new() {
    super();

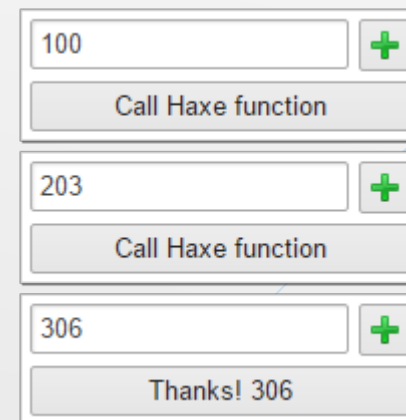
    haxeButton.onClick = function(e) {
      haxeButton.text = "Thanks! " + counterField.text;
    };

    public var startValue(null, set):Int;
    private function set_startValue(value:Int):Int {
      counterField.text = '${value}';
      return value;
    }
  }
}
```

```
Screen.instance.addComponent(new Counter());
Screen.instance.addComponent(new Counter());
Screen.instance.addComponent(new Counter());
```

Or

```
<vbox>
  <counter startValue="100" />
  <counter startValue="200" />
  <counter startValue="300" />
</vbox>
```



100	+	Call Haxe function
203	+	Call Haxe function
306	+	Thanks! 306

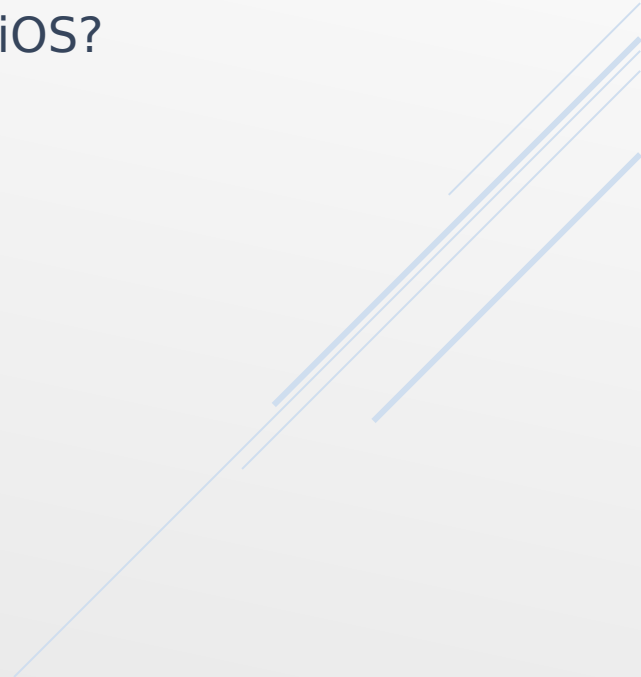
More Components

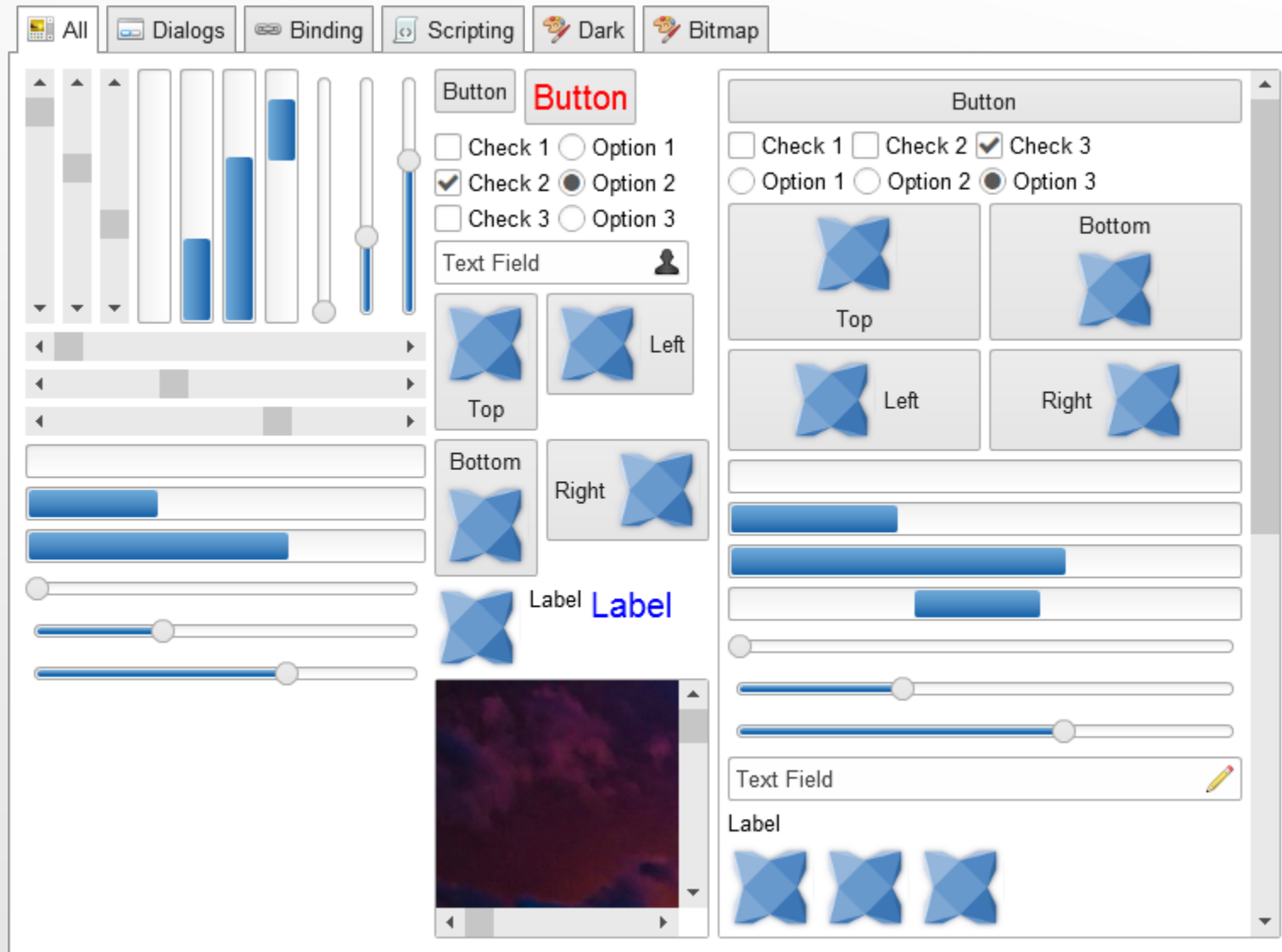
- Overlays & Dialogs
- Dropdowns
- List & ListViews
- TableViews

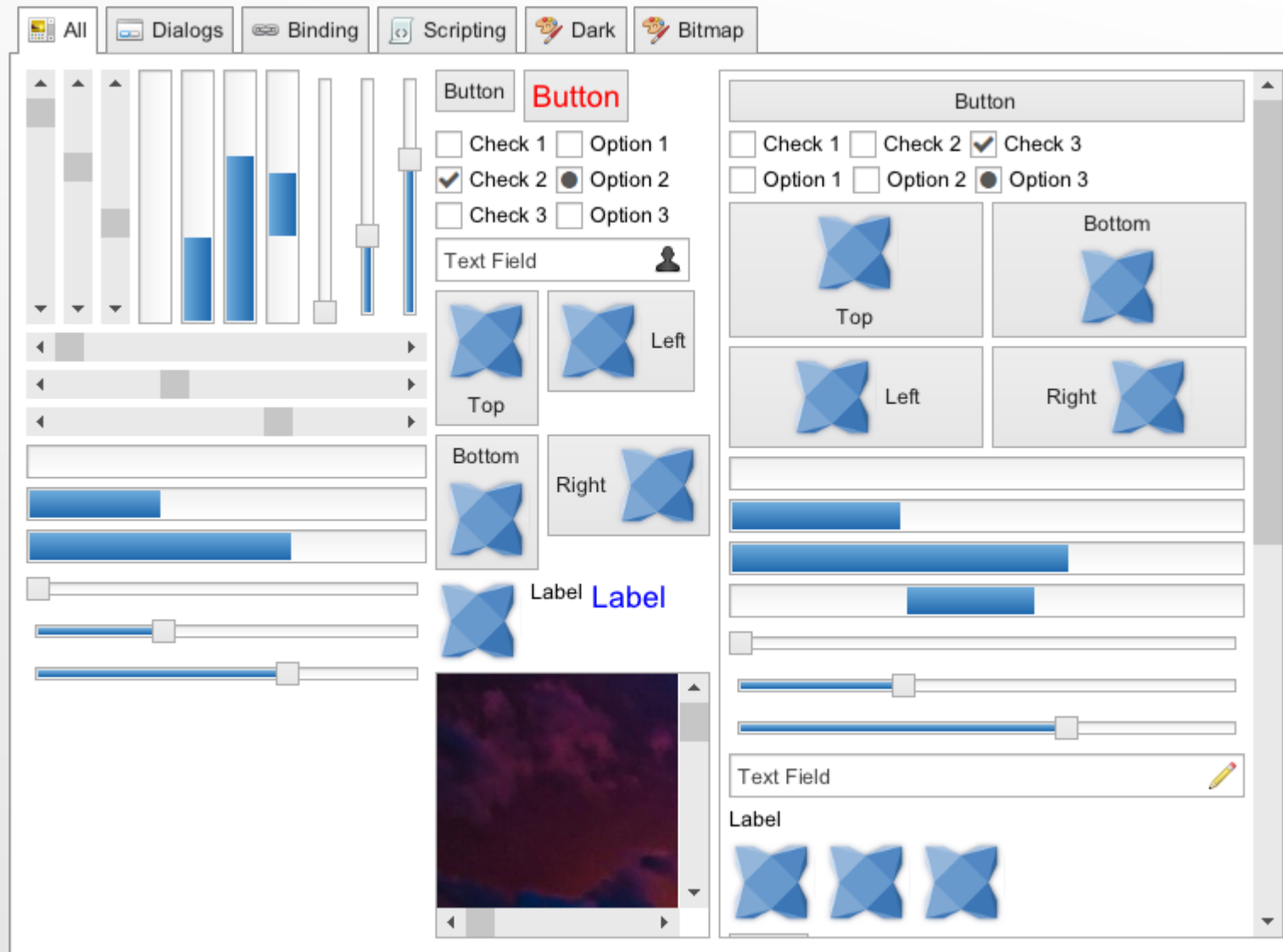
More Backends

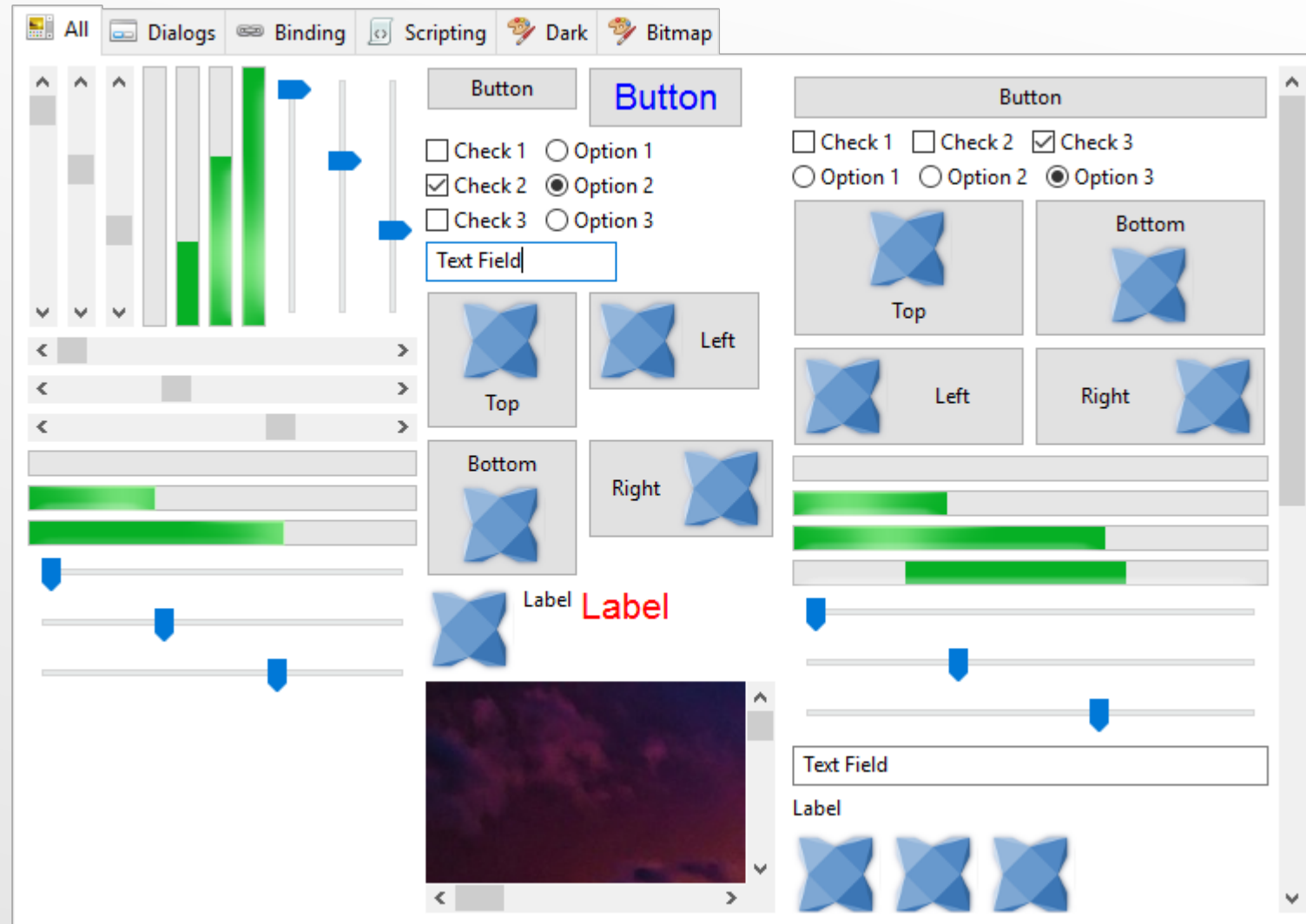
- Qt
- Heaps
- Flixel
- PDCurses?

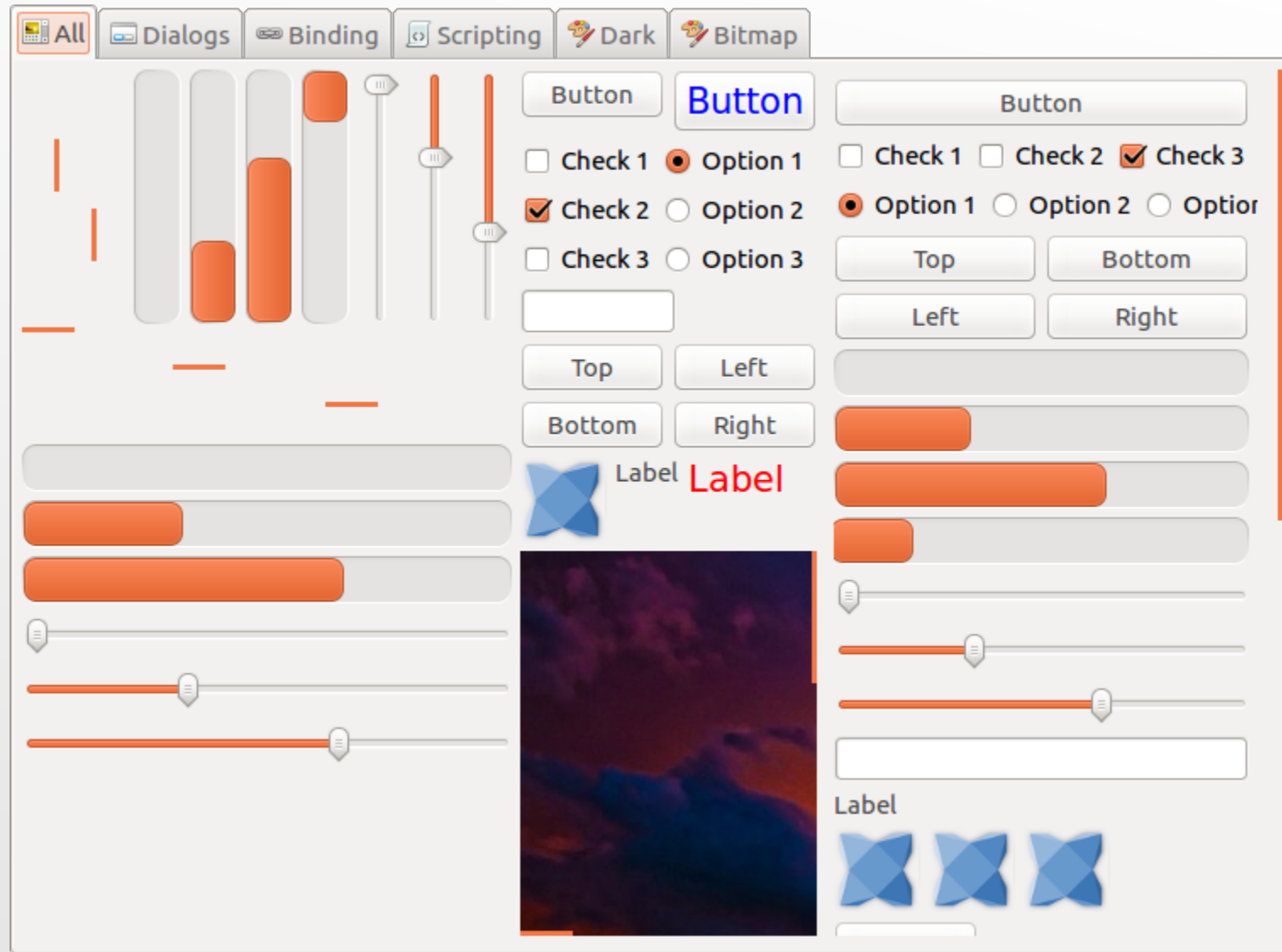
Mobile Ready

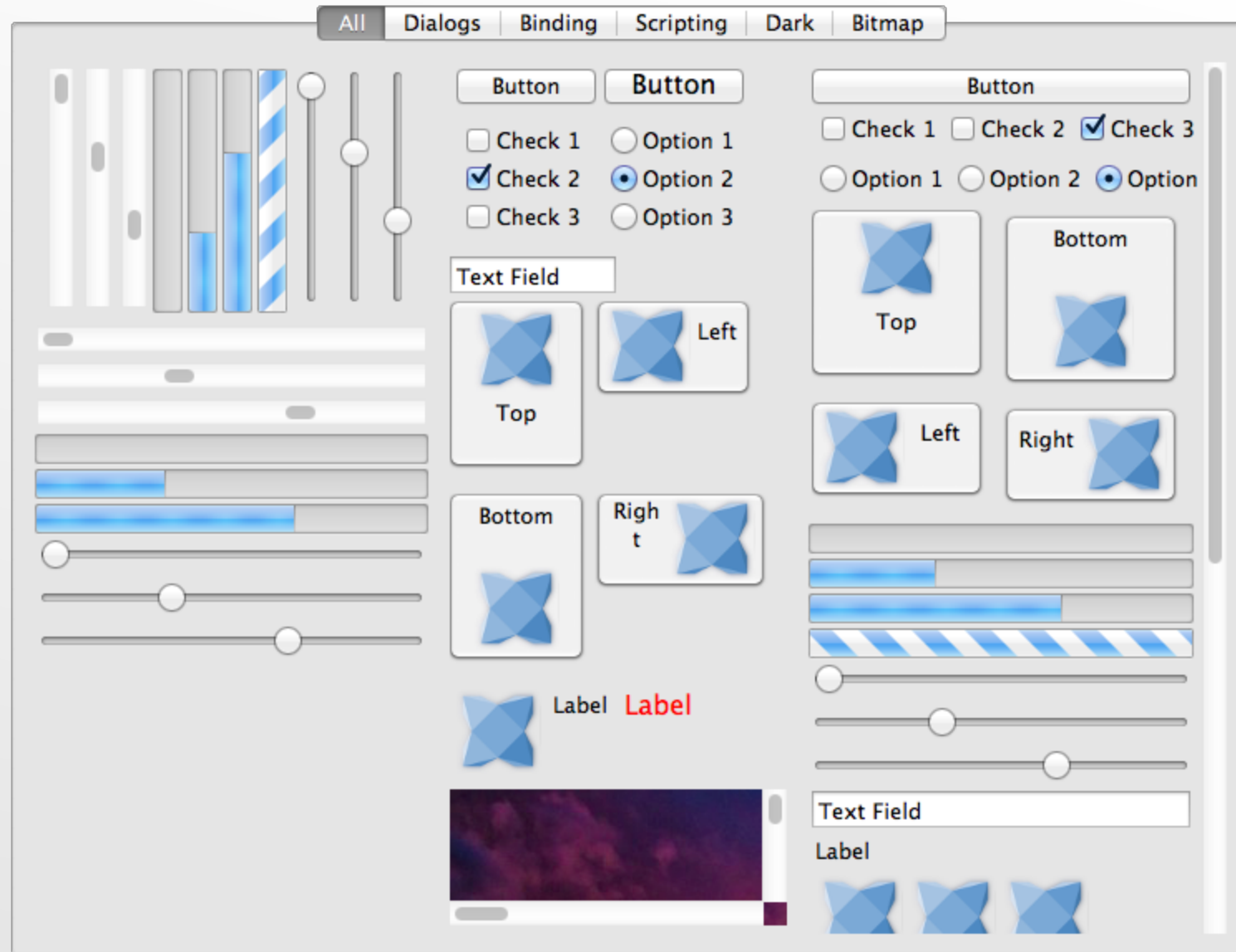
- Composites touch ready
 - DPI / Auto scaling
 - Native Android Backend
 - iOS?
- 
- Several thin, parallel blue lines of varying lengths and orientations are positioned in the bottom right corner of the slide, creating a modern, abstract design element.

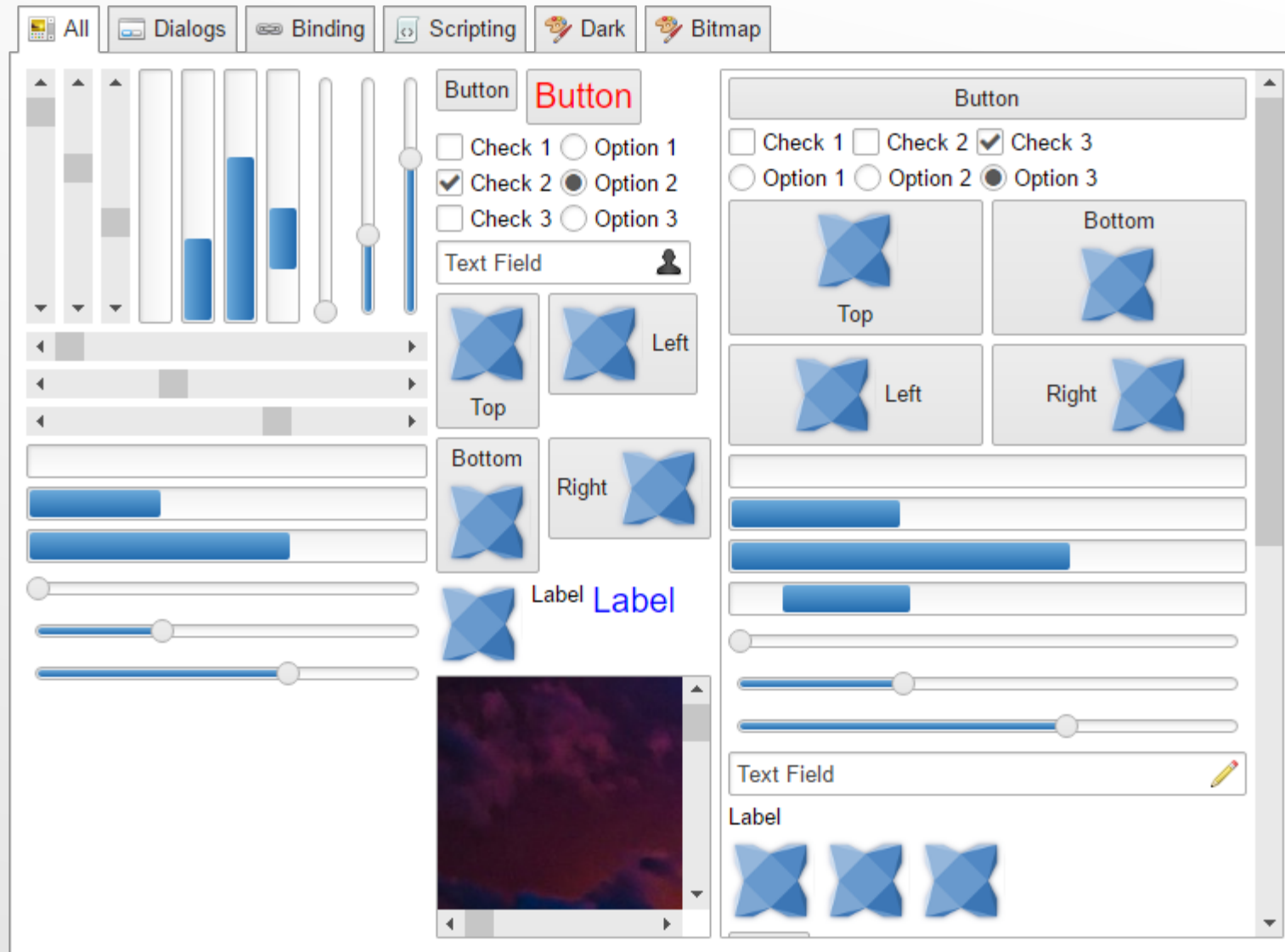


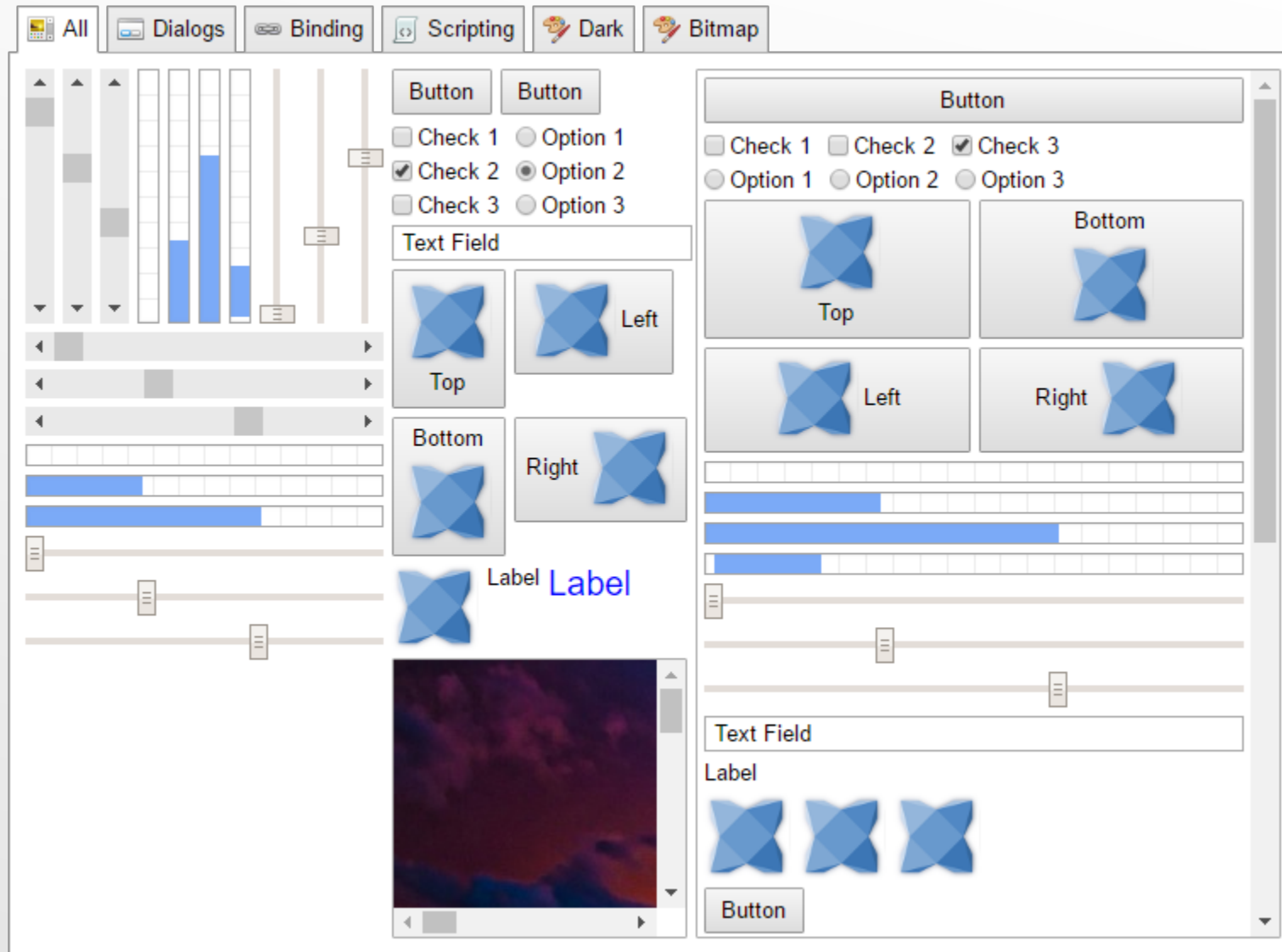


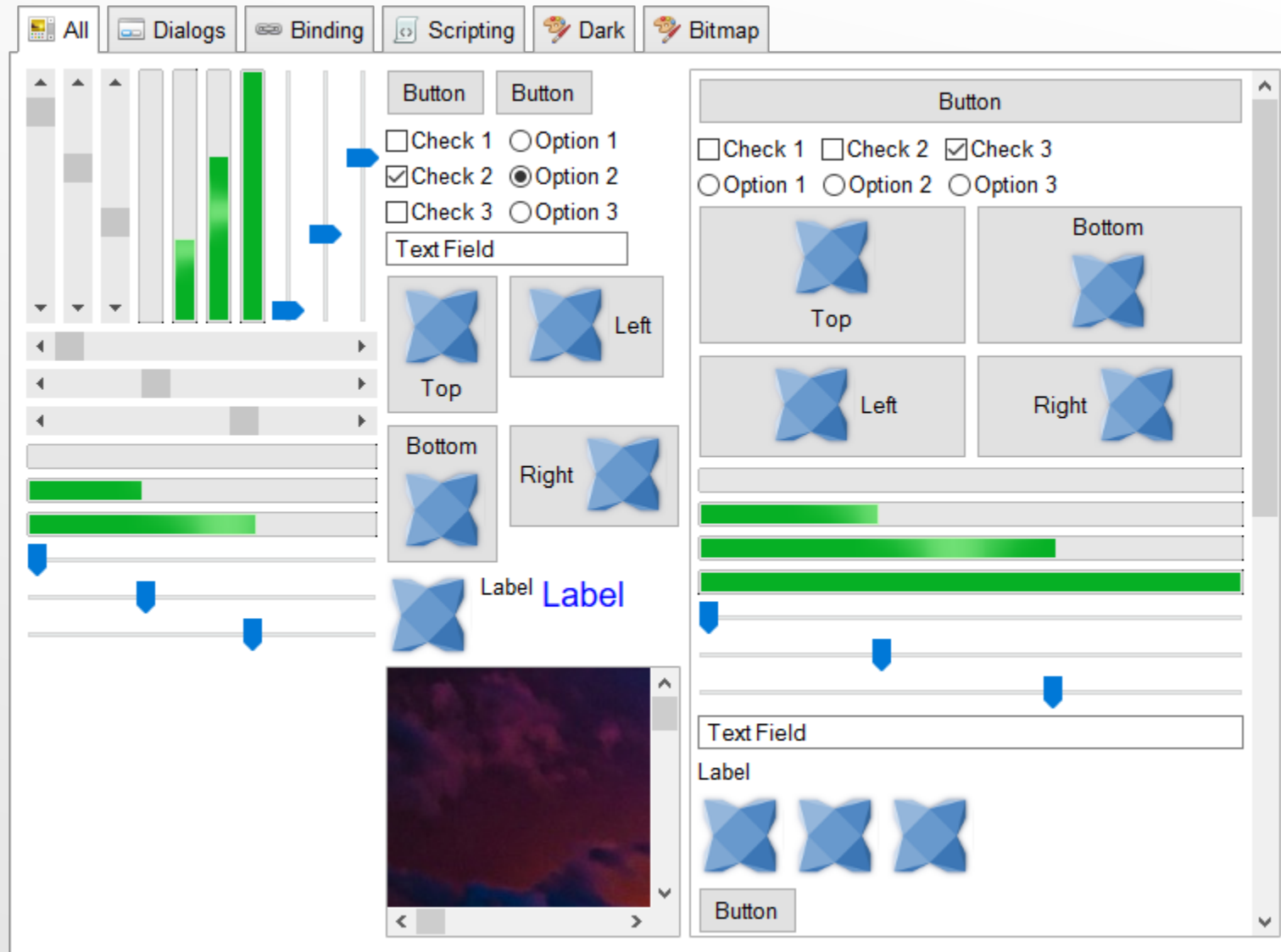


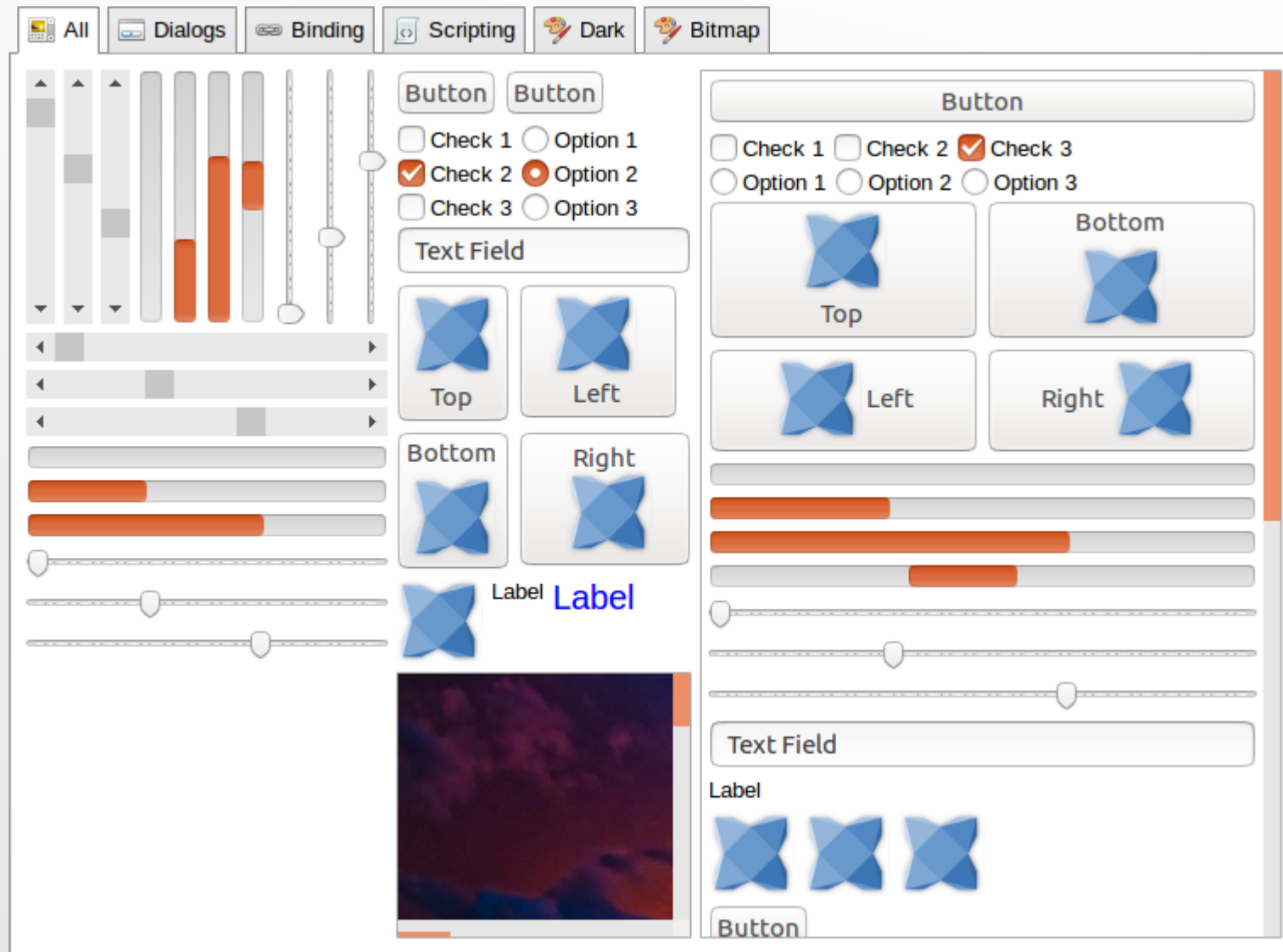


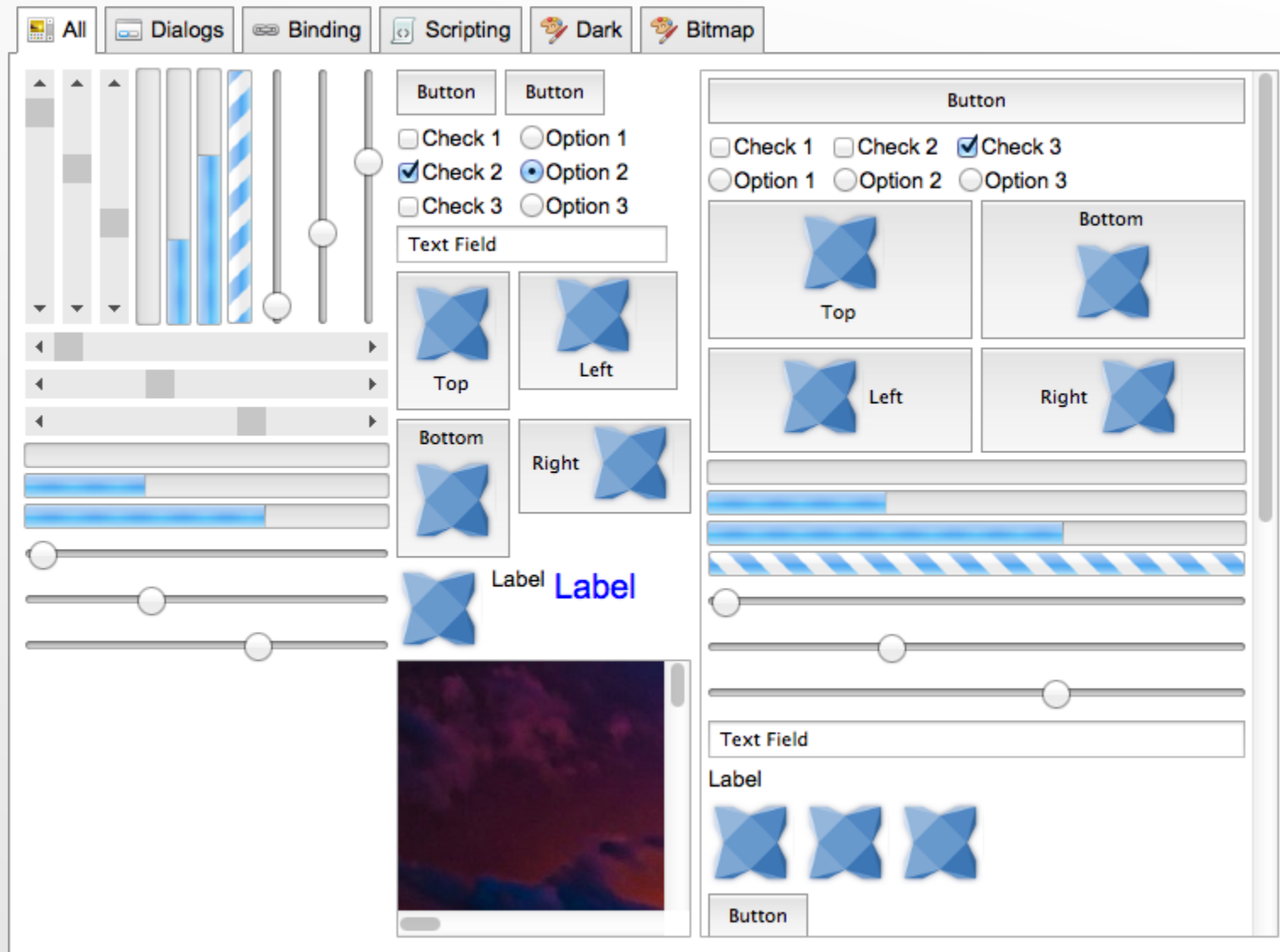


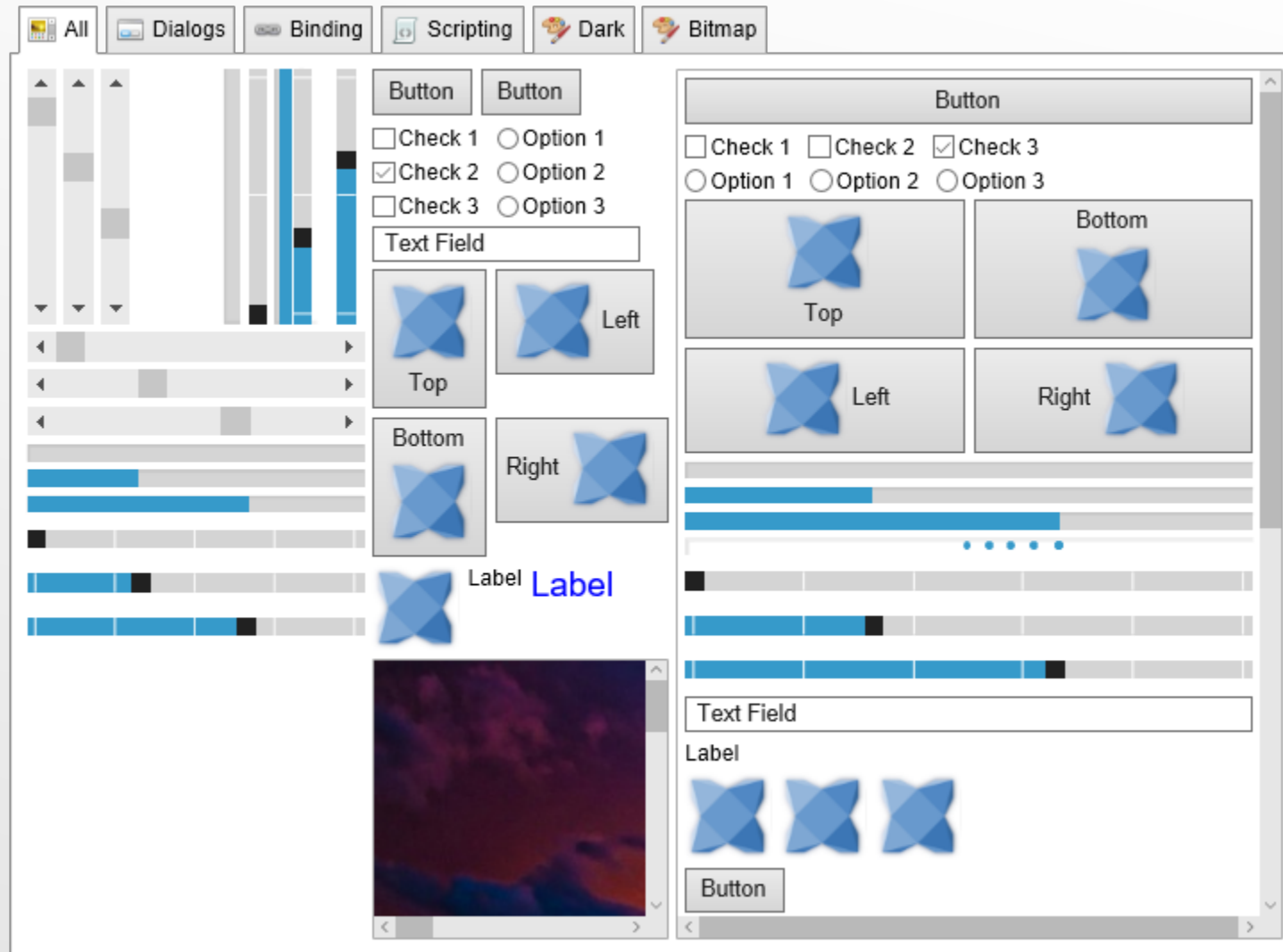












Thanks.

Several thin, parallel blue lines of varying lengths and orientations are positioned in the bottom right corner of the slide, creating a modern, abstract graphic element.