App Development Using TVMLKit

Part 2

Session 229

Jeremy Foo tvOS Engineer

Agenda

Extending Templates

Extending JavaScript

Extending Templates

MLB.com At Bat

ESPN - Get scores, n

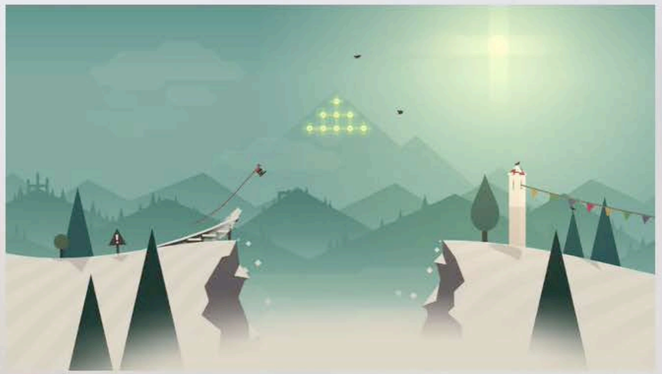
NBA 2015-16

NHL

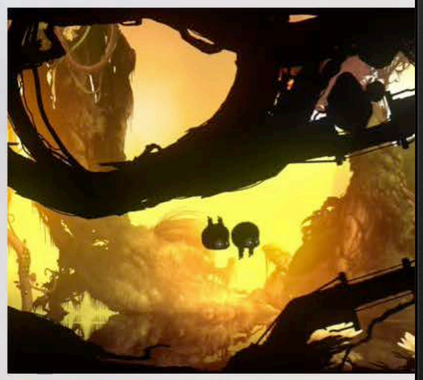
Tennis Channel Every

Essential Games

Power Hover
Games
$3.99



Alto's Adventure
Games
$3.99



BADLAND
Games

Get in Shape

TED

TED Conferences
Education 12+
★★★★★
Essentials

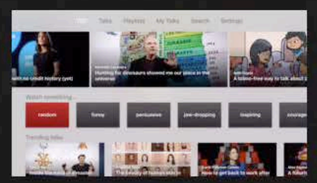
Feed your curiosity and expand your world with TED Talks.

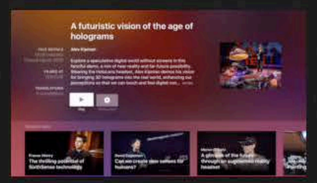
Explore more than 2,000 TED Talks from remarkable people, by topic and mood, from tech and sci... MORE

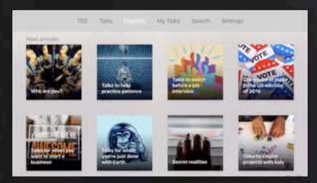
Preview

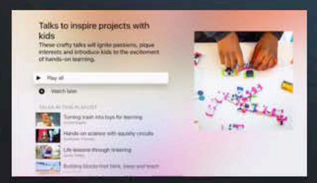
Install

Screenshots







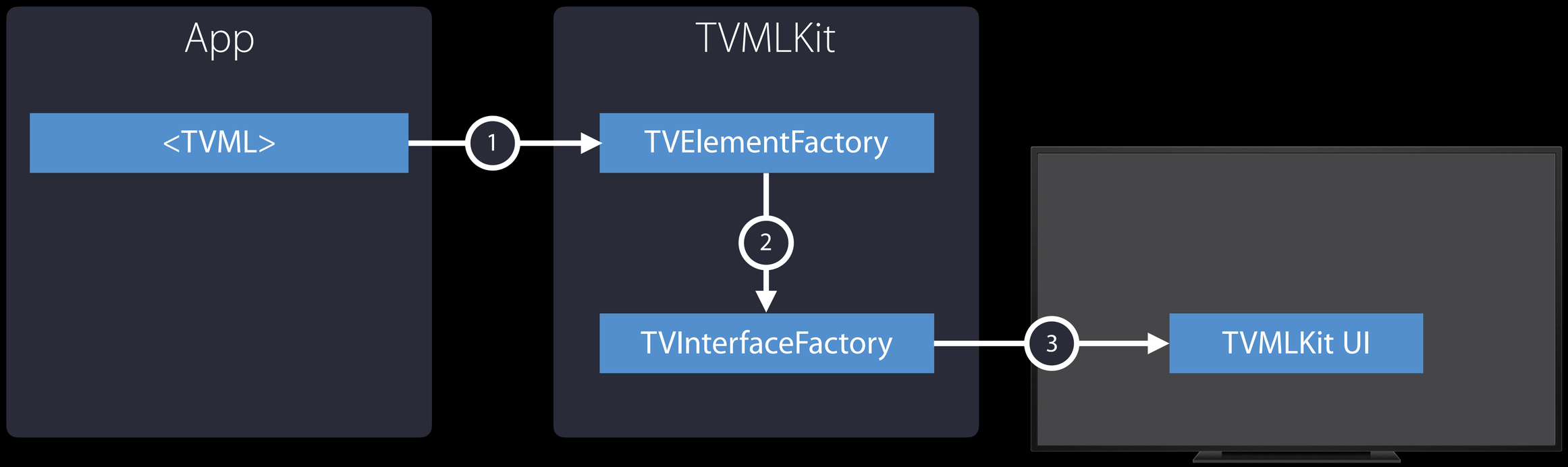


Extending Templates

App

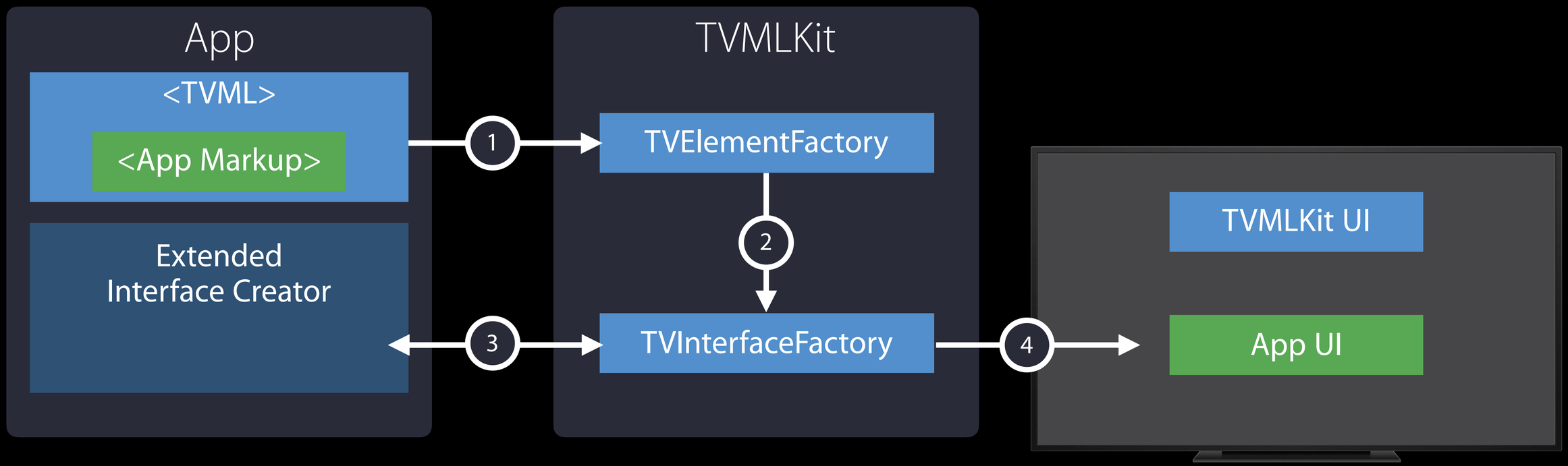
TVMLKit

Extending Templates



Extending Templates

Extending Templates



```
// XML custom banner with nested TVML button

<document>
   <stackTemplate>
      <myBanner animated="true">
         <button>...</button>
      </myBanner>
      <collectionList>
         ...
      </collectionList>
   </stackTemplate>
</document>
```

```
// XML custom banner with nested TVML button

<document>
   <stackTemplate>
      <myBanner animated="true">
         <button>...</button>
      </myBanner>
      <collectionList>
         ...
      </collectionList>
   </stackTemplate>
</document>
```

```
// XML custom banner with nested TVML button

<document>
   <stackTemplate>
      <myBanner animated="true">
         <button>...</button>
      </myBanner>
      <collectionList>
         ...
      </collectionList>
   </stackTemplate>
</document>
```

```
// XML custom banner with nested TVML button

<document>
   <stackTemplate>
      <myBanner animated="true">
         <button>...</button>
      </myBanner>
      <collectionList>
         ...
      </collectionList>
   </stackTemplate>
</document>
```

Extending Templates

Register element name

Extending Templates

Register element name

Once before app controller startup

Extending Templates

Register element name

Once before app controller startup

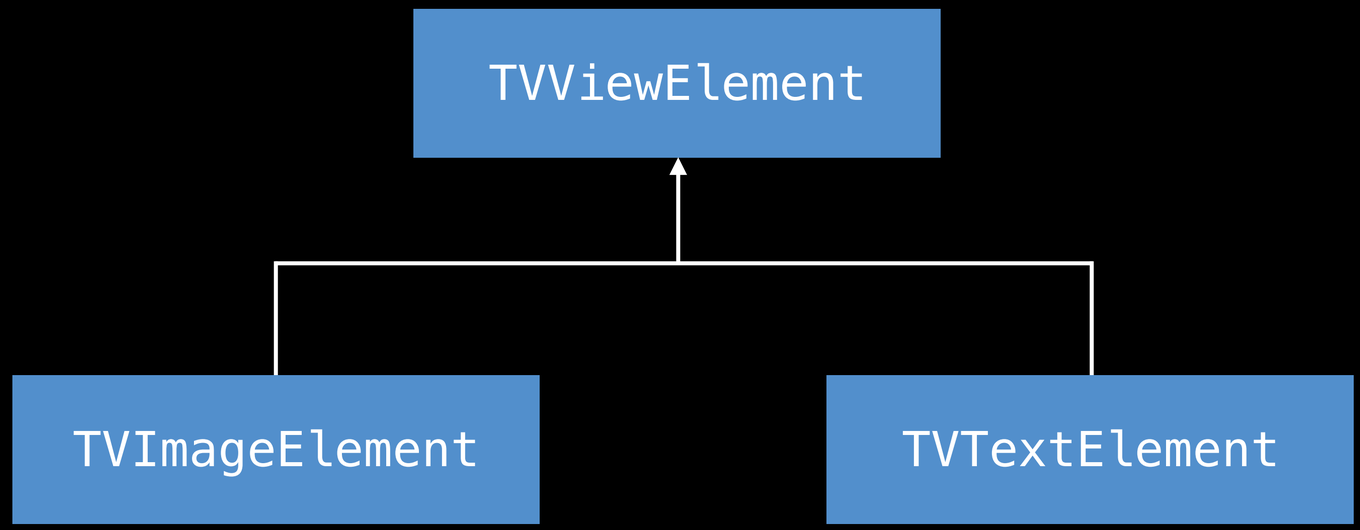
```
TVElementFactory.registerViewElementClass(TVViewElement.self, elementName: "myBanner")
```

Extending Templates

Register element name

Once before app controller startup

```
TVElementFactory.registerViewElementClass(TVViewElement.self, elementName: "myBanner")
```



Extending Templates

Interface creator

Extending Templates

Interface creator

Setup `TVInterfaceCreating` interface creator

Extending Templates

Interface creator

Setup `TVInterfaceCreating` interface creator

Configure user interface

Extending Templates

Interface creator

Setup `TVInterfaceCreating` interface creator

Configure user interface

Leverage TVMLKit

```
// Setup an interface creator
class MyInterfaceCreator: NSObject, TVInterfaceCreating {




}
```

```swift
// Setup an interface creator
class MyInterfaceCreator: NSObject, TVInterfaceCreating {
    // Conform to TVInterfaceCreating to provide extended user interface
    func makeView(element: TVViewElement, existingView: UIView?) -> UIView? {
        // code to create views
        ...
    }
}
```

```
// Setup an interface creator
class MyInterfaceCreator: NSObject, TVInterfaceCreating {
    // Conform to TVInterfaceCreating to provide extended user interface
    func makeView(element: TVViewElement, existingView: UIView?) -> UIView? {
        // code to create views
        ...
    }
}

// Register interface creator with interface factory before application controller startup
TVInterfaceFactory.shared().extendedInterfaceCreator = MyInterfaceCreator.init()
```

```
func makeView(element: TVViewElement, existingView: UIView?) -> UIView? {
    switch element.name {
        }
}
```

```
func makeView(element: TVViewElement, existingView: UIView?) -> UIView? {
    switch element.name {
    case "myBanner":
        let banner = MyBanner.init()
    }
}
```

```
func makeView(element: TVViewElement, existingView: UIView?) -> UIView? {
    switch element.name {
    case "myBanner":
        let banner = MyBanner.init()
        // Use myBanner's "animated" attribute to configure banner's animated state
        if let animated = element.attributes?["animated"] {
            banner.animated = (animated.lowercased() == "true")
        }
    }
}
```

```
func makeView(element: TVViewElement, existingView: UIView?) -> UIView? {
    switch element.name {
    case "myBanner":
        let banner = MyBanner.init()
        // Use myBanner's "animated" attribute to configure banner's animated state
        if let animated = element.attributes?["animated"] {
            banner.animated = (animated.lowercased() == "true")
        }
        // Look for button element and use TVInterfaceFactory to create the view
        var button: UIView? = nil
        if let buttonElement = self.getButtonElement(element) {
            button = TVInterfaceFactory.shared().makeView(element: buttonElement,
                existingView: button)
        }
        banner.button = button


    }
}
```

```
func makeView(element: TVViewElement, existingView: UIView?) -> UIView? {
    switch element.name {
    case "myBanner":
        let banner = MyBanner.init()
        // Use myBanner's "animated" attribute to configure banner's animated state
        if let animated = element.attributes?["animated"] {
            banner.animated = (animated.lowercased() == "true")
        }
        // Look for button element and use TVInterfaceFactory to create the view
        var button: UIView? = nil
        if let buttonElement = self.getButtonElement(element) {
            button = TVInterfaceFactory.shared().makeView(element: buttonElement,
                existingView: button)
        }
        banner.button = button
        return banner


    }
}
```

```
func makeView(element: TVViewElement, existingView: UIView?) -> UIView? {
    switch element.name {
    case "myBanner":
        let banner = MyBanner.init()
        // Use myBanner's "animated" attribute to configure banner's animated state
        if let animated = element.attributes?["animated"] {
            banner.animated = (animated.lowercased() == "true")
        }
        // Look for button element and use TVInterfaceFactory to create the view
        var button: UIView? = nil
        if let buttonElement = self.getButtonElement(element) {
            button = TVInterfaceFactory.shared().makeView(element: buttonElement,
                existingView: button)
        }
        banner.button = button
        return banner
    default:
        return nil
    }
}
```

Extending Templates

View controllers

Extending Templates

View controllers

Substitute shelf/grid type controllers

Extending Templates

View controllers

Substitute shelf/grid type controllers

Usage similar to makeView

```
func makeViewController(element: TVViewElement, existingViewController: UIViewController?)
    -> UIViewController?
```

Aha!

MLB.com At Bat

ESPN - Get scores, n

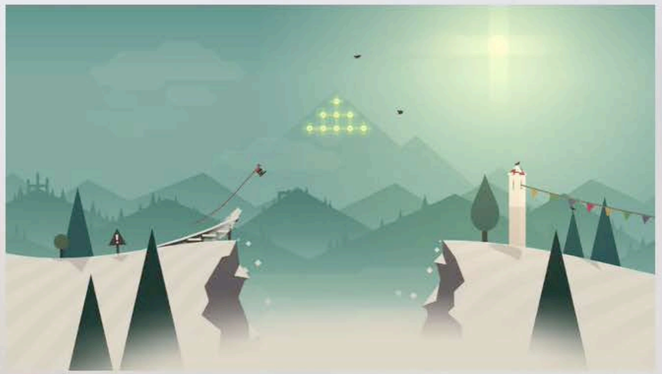
NBA 2015-16

NHL

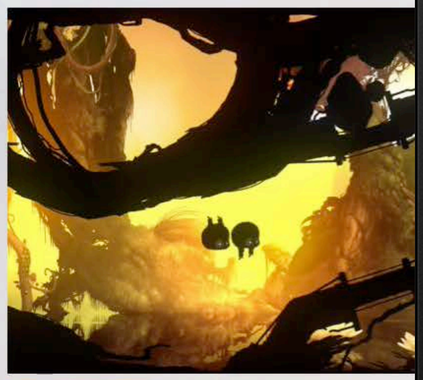
Tennis Channel Every

Essential Games

Power Hover
Games
$3.99



Alto's Adventure
Games
$3.99



BADLAND
Games

Get in Shape

Extending Templates

Custom collection view cells

NEW

Custom layout

Extending Templates

Custom collection view cells

NEW

Custom layout

Participate in focus events

Extending Templates

Custom collection view cells

NEW

Custom layout

Participate in focus events

```
func collectionViewCellClass(for element: TVViewElement) -> AnyClass?
```

Extending Templates

Custom collection view cells

NEW

Custom layout

Participate in focus events

```
func collectionViewCellClass(for element: TVViewElement) -> AnyClass?
```

```
func makeView(element: TVViewElement, existingView: UIView?) -> UIView?
```

Demo

Custom collection view cell

Parry Panesar tvOS Engineer

Recap

Define custom markup

Recap

Define custom markup

Register custom elements

Recap

Define custom markup

Register custom elements

Provide extended interface creator

Recap

Define custom markup

Register custom elements

Provide extended interface creator

Configure custom user interface

Best Practices

Handling Document Updates

Updated anytime

Handling Document Updates

Updated anytime

Check update type

```
switch element.updateType {
   case .node:
      // Update current element and children
      break
   case .subtree:
      // Update children
      break
   case .children:
      // Update children with changed order
      break
   default:
      break
}
```

Handling Document Updates

Updated anytime

Check update type

Reuse views

```
switch element.updateType {
   case .node:
      // Update current element and children
      break
   case .subtree:
      // Update children
      break
   case .children:
      // Update children with changed order
      break
   default:
      break
}
```

```
func makeView(element: TVViewElement, existingView: UIView?) -> UIView? {
    switch element.name {
    case "myBanner":
        let banner = MyBanner.init()
        // Use myBanner's "animated" attribute to configure banner's animated state
        if let animated = element.attributes?["animated"] {
            banner.animated = (animated.lowercased() == "true")
        }
        // Look for button element and use TVInterfaceFactory to create the view
        var button: UIView? = nil
        if let buttonElement = self.getButtonElement(element) {
            button = TVInterfaceFactory.shared().makeView(element: buttonElement,
                existingView: button)
        }
        banner.button = button
        return banner
    default:
        return nil
    }
}
```

```
func makeView(element: TVViewElement, existingView: UIView?) -> UIView? {
    switch element.name {
    case "myBanner":
        let banner = (existingView as? MyBanner) ?? MyBanner.init()
        // Use myBanner's "animated" attribute to configure banner's animated state
        if let animated = element.attributes?["animated"] {
            banner.animated = (animated.lowercased() == "true")
        }
        // Look for button element and use TVInterfaceFactory to create the view
        var button: UIView? = banner.button
        if let buttonElement = self.getButtonElement(element) {
            button = TVInterfaceFactory.shared().makeView(element: buttonElement,
                existingView: button)
        }
        banner.button = button
        return banner
    default:
        return nil
    }
}
```

```
func makeView(element: TVViewElement, existingView: UIView?) -> UIView? {
    switch element.name {
    case "myBanner":
        let banner = (existingView as? MyBanner) ?? MyBanner.init()
        // Use myBanner's "animated" attribute to configure banner's animated state
        if let animated = element.attributes?["animated"] {
            banner.animated = (animated.lowercased() == "true")
        }
        // Look for button element and use TVInterfaceFactory to create the view
        var button: UIView? = banner.button
        if let buttonElement = self.getButtonElement(element) {
            button = TVInterfaceFactory.shared().makeView(element: buttonElement,
                existingView: button)
        }
        banner.button = button
        return banner
    default:
        return nil
    }
}
```

```
func makeView(element: TVViewElement, existingView: UIView?) -> UIView? {
    switch element.name {
    case "myBanner":
        let banner = (existingView as? MyBanner) ?? MyBanner.init()
        // Use myBanner's "animated" attribute to configure banner's animated state
        if let animated = element.attributes?["animated"] {
            banner.animated = (animated.lowercased() == "true")
        }
        // Look for button element and use TVInterfaceFactory to create the view
        var button: UIView? = banner.button
        if let buttonElement = self.getButtonElement(element) {
            button = TVInterfaceFactory.shared().makeView(element: buttonElement,
                existingView: button)
        }
        banner.button = button
        return banner
    default:
        return nil
    }
}
```

Adapting to Appearance

Custom user interface

Listen to trait collection changes

Adapting to Appearance

Custom user interface

Listen to trait collection changes

Adapting to Appearance

TVML components

Check style update type

```
switch element.updateType {
    ...
    case .styles:
        // update styles based on new styles
        break
   default:
        break
}
```

Adapting to Appearance

TVML components

Check style update type

Must reuse components

```
switch element.updateType {
    ...
    case .styles:
        // update styles based on new styles
        break
   default:
        break
}
```

Adapting to Appearance

TVML components

Check style update type

Must reuse components

Must forward to TVInterfaceFactory

```
switch element.updateType {
    ...
    case .styles:
        // update styles based on new styles
        break
   default:
        break
}
```

Mix Native Controller

Define custom template element

Mix Native Controller

Define custom template element

```
// Register an element for your view controller
TVElementFactory.registerViewElementClass(TVViewElement.self, elementName: "myViewController")

// Vend your view controller
func makeViewController(element: TVViewElement, existingViewController: UIViewController?) ->
   UIViewController? {
   switch element.name {
      case "myViewController":
         return MyViewController.init(/* initialization */)
      default:
         return nil
   }
}
```

Mix Native Controller

Return your view controller

```
// Register an element for your view controller
TVElementFactory.registerViewElementClass(TVViewElement.self, elementName: "myViewController")

// Vend your view controller
func makeViewController(element: TVViewElement, existingViewController: UIViewController?) ->
   UIViewController? {
   switch element.name {
      case "myViewController":
         return MyViewController.init(/* initialization */)
      default:
         return nil
   }
}
```

Sub Application

Host the navigationController

Sub Application

Host the navigationController

```
// Create hosted controller
let hostedControllerContext = TVApplicationControllerContext()
hostedControllerContext.javaScriptApplicationURL = javaScriptURL
let hostedController = TVApplicationController(context: hostedControllerContext,
    window: nil, delegate: self)

// Present hosted controller
let navigationController = hostedController.navigationController
self.present(navigationController, animated: true, completion: nil)
```

Sub Application

Host the navigationController

Host in separate UIWindow

Extending JavaScript

Christopher Bonhage tvOS Engineer

Extending JavaScript

JavaScript libraries

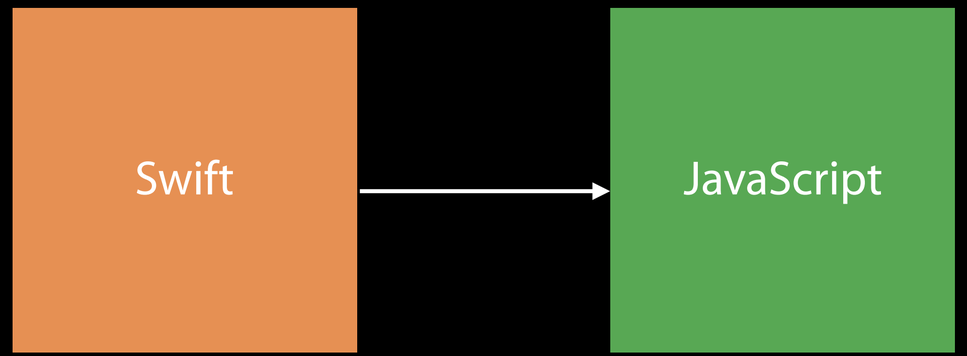
Extending JavaScript

JavaScript libraries

Extending JavaScript

JavaScript libraries

Calling into JavaScript



Extending JavaScript

JavaScript libraries

Calling into JavaScript

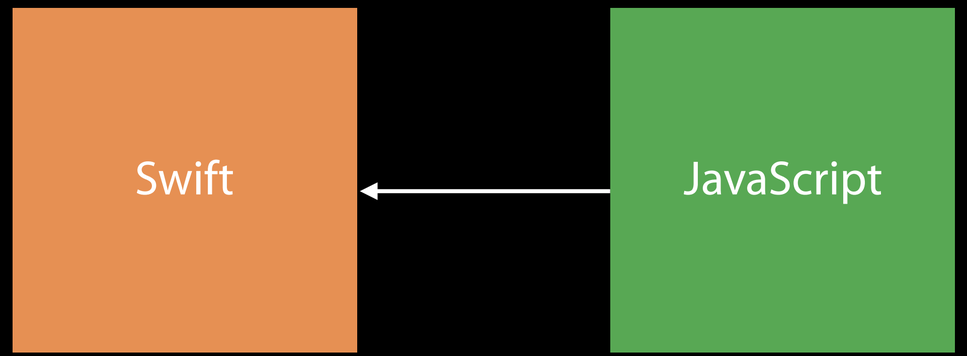
Bridging to JavaScript

Extending JavaScript

JavaScript libraries

Calling into JavaScript

Bridging to JavaScript



JavaScript Libraries

JavaScript Libraries

Load additional scripts

Executes in the global context

JavaScript Libraries

Load additional scripts

Executes in the global context

JavaScript Libraries

Load additional scripts

Executes in the global context

```
const scriptURLs = [
    options.BASEURL + "js/DocumentLoader.js",
    options.BASEURL + "js/DocumentController.js"
];
evaluateScripts(scriptURLs, function(scriptsAreLoaded) {
    // Continue with App.onLaunch
});
```

JavaScript Libraries

Caveats

JavaScript Libraries

Caveats

Evaluate once

JavaScript Libraries

Caveats

Evaluate once 

All-or-nothing 

JavaScript Libraries

Caveats

Evaluate once 

All-or-nothing 

Not a web browser 

Calling into JavaScript

Calling into JavaScript

Request the JSContext

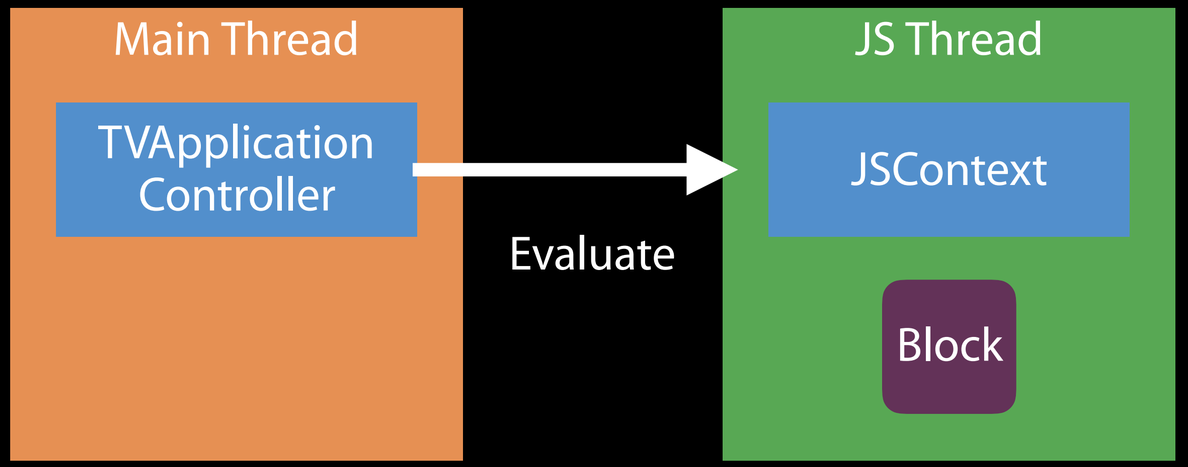
Calling into JavaScript

Request the JSContext

Calling into JavaScript

Request the JSContext

Evaluate with context

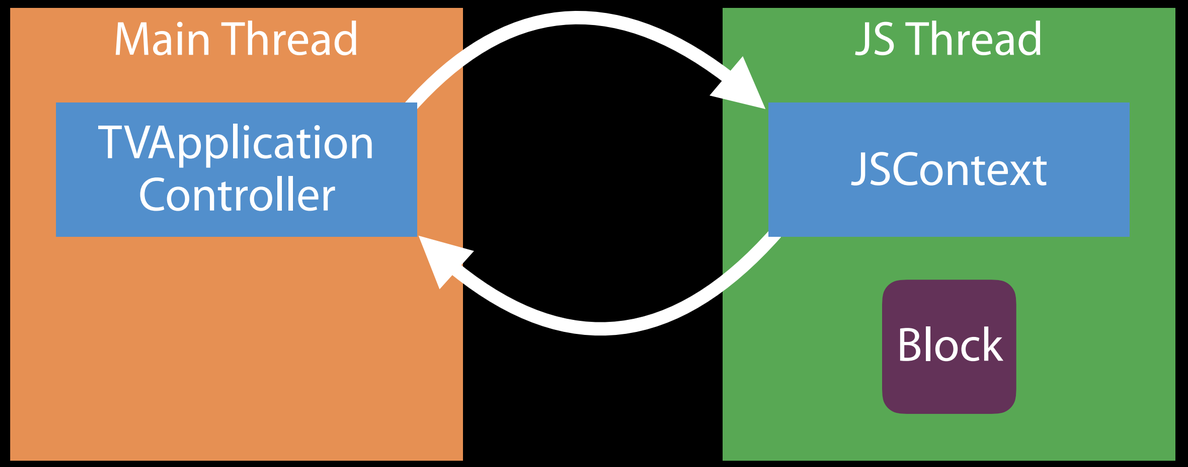


Calling into JavaScript

Request the JSContext

Evaluate with context

Don’t block main thread

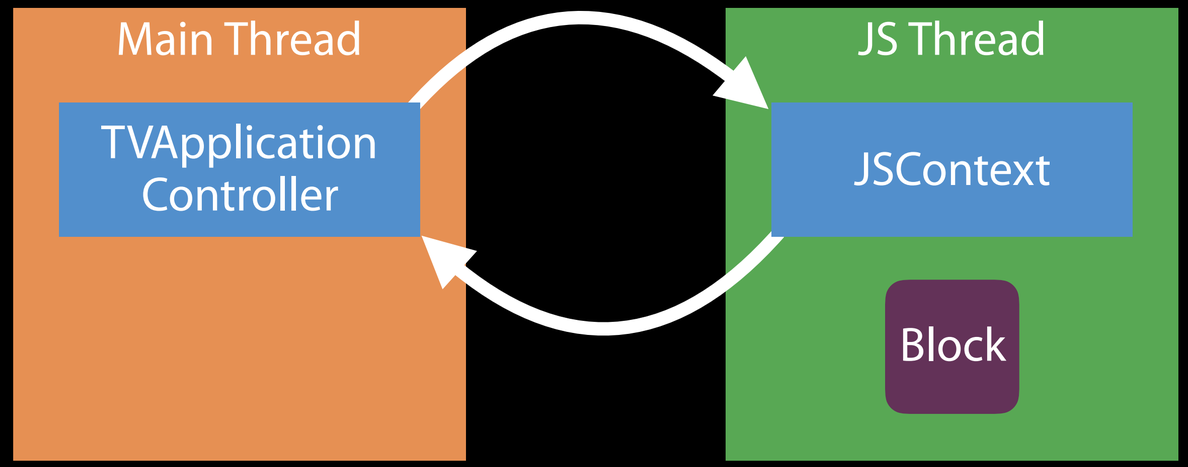


Calling into JavaScript

Request the JSContext

Evaluate with context

Don’t block main thread



```
// Calling into JavaScript example

func application(_ app: UIApplication, open url: URL, options: [String : AnyObject] = [:])
      -> Bool {

    return true
}
```

```
// Calling into JavaScript example

func application(_ app: UIApplication, open url: URL, options: [String : AnyObject] = [:])
      -> Bool {

    return true
}
```

```swift
// Calling into JavaScript example

func application(_ app: UIApplication, open url: URL, options: [String : AnyObject] = [:])
      -> Bool {
   // Request the context
   appController?.evaluate(inJavaScriptContext: { (context) in

   }, completion: nil)
   return true
}
```

```swift
// Calling into JavaScript example

func application(_ app: UIApplication, open url: URL, options: [String : AnyObject] = [:])
      -> Bool {
    // Request the context
    appController?.evaluate(inJavaScriptContext: { (context) in
    }, completion: nil)
    return true
}
```

```swift
// Calling into JavaScript example

func application(_ app: UIApplication, open url: URL, options: [String : AnyObject] = [:])
      -> Bool {
   // Request the context
   appController?.evaluate(inJavaScriptContext: { (context) in
      // Evaluate in context
      if context.globalObject.hasProperty("onOpenURL") {
         let urlString = url.absoluteString as AnyObject
         context.globalObject.invokeMethod("onOpenURL", withArguments: [urlString])
      }
   }, completion: nil)
   return true
}
```

```swift
// Calling into JavaScript example

func application(_ app: UIApplication, open url: URL, options: [String : AnyObject] = [:])
      -> Bool {
   // Request the context
   appController?.evaluate(inJavaScriptContext: { (context) in
      // Evaluate in context
      if context.globalObject.hasProperty("onOpenURL") {
         let urlString = url.absoluteString as AnyObject
         context.globalObject.invokeMethod("onOpenURL", withArguments: [urlString])
      }
   }, completion: nil)
   return true
}
```

Bridging to JavaScript

Bridging to JavaScript

Create custom protocol

Bridging to JavaScript

Create custom protocol

Implement in Objective-C class

Bridging to JavaScript

Create custom protocol

Implement in Objective-C class

Expose via AppDelegate

```
// Bridging to JavaScript example
```

```
// Bridging to JavaScript example
```

```
// Bridging to JavaScript example

@objc protocol StoreKitWrapperProtocol : JSExport {
  // Definition of custom protocol
}
```

```
// Bridging to JavaScript example

@objc protocol StoreKitWrapperProtocol : JSExport {
  // Definition of custom protocol
}
```

```
// Bridging to JavaScript example

@objc protocol StoreKitWrapperProtocol : JSExport {
  // Definition of custom protocol
}

class StoreKitWrapper: NSObject, StoreKitWrapperProtocol {
  // Implementation of custom protocol
}
```

```
// Bridging to JavaScript example

@objc protocol StoreKitWrapperProtocol : JSExport {
  // Definition of custom protocol
}

class StoreKitWrapper: NSObject, StoreKitWrapperProtocol {
  // Implementation of custom protocol
}
```

```
// Bridging to JavaScript example

@objc protocol StoreKitWrapperProtocol : JSExport {
  // Definition of custom protocol
}

class StoreKitWrapper: NSObject, StoreKitWrapperProtocol {
  // Implementation of custom protocol
}
```

```
func appController(appController: TVApplicationController,
      evaluateAppJavaScriptInContext context: JSContext) {
   context.setObject(StoreKitWrapper.self, forKeyedSubscript: "StoreKitWrapper")
}
```

```
// Bridging to JavaScript example

@objc protocol StoreKitWrapperProtocol : JSExport {
  // Definition of custom protocol
}

class StoreKitWrapper: NSObject, StoreKitWrapperProtocol {
  // Implementation of custom protocol
}
```

```
func appController(appController: TVApplicationController,
      evaluateAppJavaScriptInContext context: JSContext) {
   context.setObject(StoreKitWrapper.self, forKeyedSubscript: "StoreKitWrapper")
}
```

Summary

Easiest way to provide custom user experiences

Build on top of existing TVMLKit capabilities

Create unique, immersive apps

More Information

https://developer.apple.com/wwdc16/229

Related Sessions

Designing for tvOS	Presidio	Tuesday 2:00PM
Mastering UIKit for tvOS	Presidio	Wednesday 10:00AM
Developing tvOS Apps Using TVMLKit: Part 1	Mission	Wednesday 1:40PM
Focus Interactions on tvOS	Mission	Wednesday 4:00PM
Optimizing Web Content in Your App	Mission	Friday 4:00PM

Labs

TVMLKit Lab	Graphics, Games, and Media Lab C	Friday 9:00AM

WWDC16