Sample Code

Creating a Client-Server TVML App

Display and navigate between TVML documents on Apple TV by retrieving and parsing information from a remote server.

Download

tvOS 12.1+ | Xcode 11.3+

Overview

A TVML app creates a client-server connection to retrieve information stored on a server. The retrieved information is parsed into a document and displayed on a TV screen. Use this sample code project to create your first client-server app. The app uses JavaScript to load an initial TVML document from a local server. The user navigates between two images and the app loads a new document after the user selects one of the images. For detailed information about the TVML framework, see TVML.

Configure the Sample Code Project

Before running the app, you need to set up a local server on your machine:

1. In Finder, navigate to the CreateTVMLApp directory inside of the CreateTVMLApp project directory.
2. In Terminal, enter at the prompt, `cd` followed by a space.
3. Drag the CreateTVMLApp folder from the Finder window into the Terminal window, and press Return. This changes the directory to that folder.
4. In Terminal, enter `ruby -run -ehttpd . -p9001` to run the server.
5. Build and run the app.

After testing the sample app in Apple TV Simulator, you can close the local server by pressing Control-C in Terminal. Closing the Terminal window also kills the server.

Display the Initial Document

The `application.js` file controls the app. The app creates two global variables that contain URL information. When the app launches, the app populates the variables with information contained in the `AppDelegate.swift` file and retrieves the first document from the server.

```
var baseURL;
var serverURL;

App.onLaunch = function(options) {
    baseURL = options.BASEURL;
    serverURL = options.BASEURL + "/Server";
    var extension = "/Templates/InitialPage.xml";
    getDocument(extension);
}
```

Retrieve a TVML Document From the Server

While you can't control a user's internet access, you can control what they see on the screen. To avoid showing a blank screen, create and display a loading document to provide users with a visual cue that your app is working, despite not having a connection to the server.

Create and display the loading document from inside your main JavaScript file. This ensures that you can always display the loading document, even if access to the server is down. The following code creates a loading document and pushes it onto the navigation stack for display.

```
function loadingTemplate() {
    var template = '<document><loadingTemplate><activityIndicator><text>Loading</te
    var templateParser = new DOMParser();
    var parsedTemplate = templateParser.parseFromString(template, "application/xml")
    navigationDocument.pushDocument(parsedTemplate);
    return parsedTemplate;
}
```

Create a new `XMLHttpRequest` to retrieve information from the server. After successfully loading a new document, push the document onto the navigation stack to display it.

```
function getDocument(extension) {
    var templateXHR = new XMLHttpRequest();
    var url = serverURL + extension;
    var loadingScreen = loadingTemplate();

    templateXHR.responseType = "document";
    templateXHR.addEventListener("load", function() {pushPage(templateXHR.responseXM
    templateXHR.open("GET", url, true);
    templateXHR.send();
}
```

Replace the Previous Document

After the user selects a new document, push that document onto the navigation stack. This places the new document at the top of the current document stack and displays it. When the user presses the Menu button on the Siri Remote, the system removes the current document from the stack and displays the previous document. Doing this causes the previous loading document to display when you want the user to see the selection document.

To fix this, replace the loading document with the new document using the `replaceDocument` method. The following function takes the new document and the loading document as parameters and replaces the loading document with the new document.

```
function pushPage(page, loading) {
    var currentDoc = getActiveDocument();
    navigationDocument.replaceDocument(page, loading);
}
```

Select a New TVML Document

Each file on your server contains the information to display a single document onscreen. In this app, the initial document contains information that displays two images, each with their own title. Each image is a Lockup Elements element. When the user selects a lockup, the app uses the `onselect` attribute to call the `getDocument` method. The lockup passes the URL for the next document to the `getDocument` method.

```
<lockup onselect="getDocument('/Templates/PageOne.xml')">
```

The image element contains the URL and size specifications for the displayed image.

```
<img width="182" height="274" src="http://localhost:
9001/Server/Images/Car_Movie_250x375_A.png"/>
```