Multi-Screen SDK

Published 2013-12-26 | Supported SDK version 5.0| Supported Platform 14TV

Overview

Create compelling Multi-Screen applications that extend your mobile application experience to the TV.

The Multi-Screen SDK provides developers with a set of API's and tools that enable multi-screen experiences for Games, Media Sharing, Social Collaboration and more.

A Multi-Screen application is divided into two applications. One application runs on a Samsung SmartTV, and the other application runs on a mobile device. The Multi-Screen SDK provides an API for your TV Application and APIs for Android, iOS, and Javascript mobile applications.

The Multi-Screen API's focus on 4 main features:

1) Discover
   The mobile APIs provide the ability to discover a compatible Samsung TV from your mobile application.

2) Launch
   Once a TV is discovered, you can launch your TV application directly from the mobile API.

3) Connect
   Once the TV application is launched, your mobile application can connect to your TV application. Connections are based on web sockets, and provide bi-directional, reliable, and fast communication.

4) Communicate
   Once connected, each device has awareness of all other devices connected. This allows one device to send messages to any other device, or group of devices, including the TV.

Prerequisites

To get started with the Multi-Screen SDK, you will need:

- A pc or mac connected to the internet.
- For developing TV applications, you will need a text editor or suitable IDE for editing HTML, Javascript, and CSS files.
- For developing Android mobile applications, you will need Eclipse or Android Studio with the Android SDK installed.
- For developing iOS mobile applications, you will need a mac running Xcode and an iOS developer account.

SmartTV SDK and Emulator

To run the Examples and test your application, you will need to install the Samsung SmartTV SDK and Emulator. Please refer to these instructions for download and installation.

There are two key settings to modify in the Emulator before developing your multi-screen application.

1) Bridged Networking
   In order for your mobile device to "discover" the emulator, you will need to modify the VirtualBox Network settings, and attach to the "Bridged Adapter".

2) Shared Folders
   In the Shared Folders setting, edit the "Apps" path to point to a directory on your machine. This is where you can place TV applications that can run in the Emulator.

Download Multi-Screen SDK

Download the Multi-Screen API's and examples. Each API download contains libs, docs, and examples.

http://multiscreen.samsung.com/downloads/MultiScreenSDK-1.1.11.zip

SmartTV Application API

For SmartTV Applications, the following javascript file is required:
```
<script type="text/javascript" src="multiscreen-smarttv-1.1.11.min.js"></script>
```

Mobile Application APIs

1) Android API
   multiscreen-android-1.1.11.jar *is required for Android mobile applications.*

2) iOS API
   SamsungMultiscreen.framework *is required for iOS mobile applications.*

3) Javascript API
   For mobile javascript applications, the following script is required:
   ```
   <script type="text/javascript" src="multiscreen-mobile-1.1.11.min.js"></script>
   ```

Running the Sample Application

To verify that your setup is working correctly, run the sample TV application and sample mobile application.

Sample TV Application
First, open the sample TV application in the emulator.

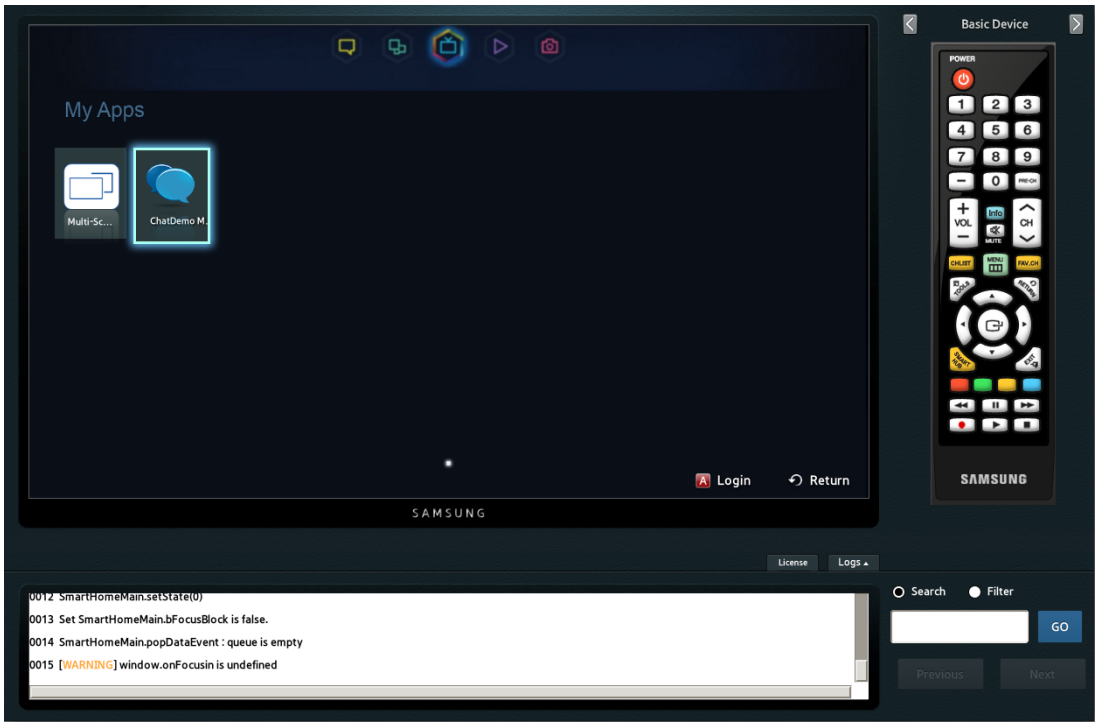
1) Open the **examples/ChatDemo/tv/** folder

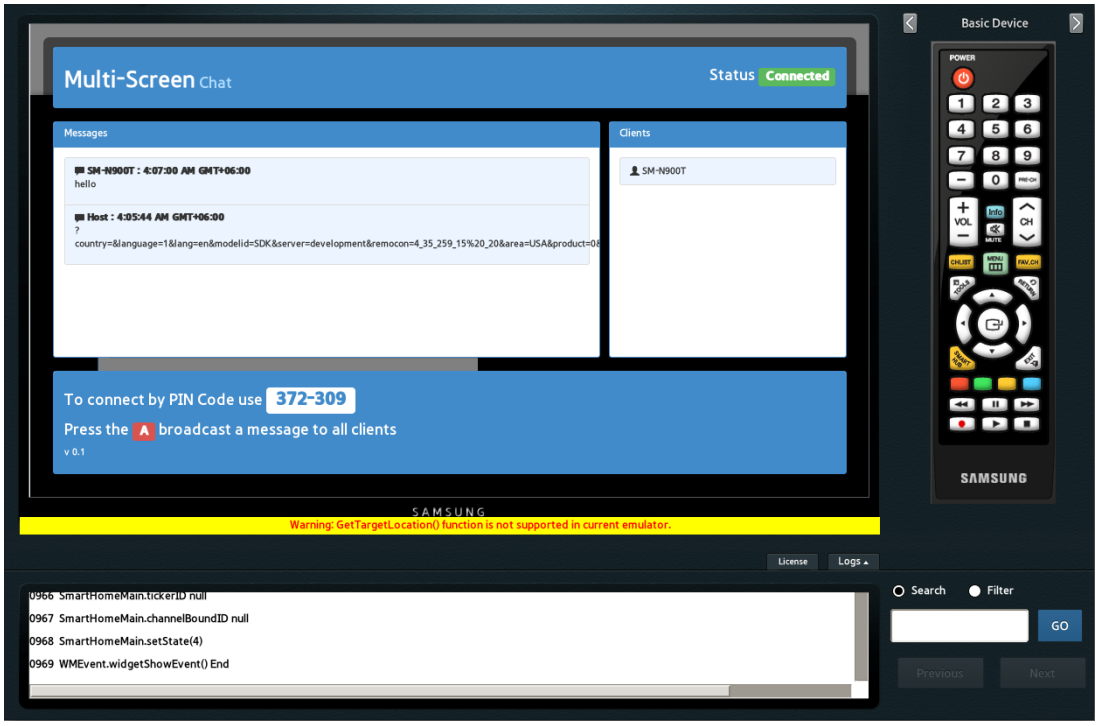
2) Copy the **chatdemo-tv** directory to the directory on your machine that is mapped as the shared "Apps" folder for the emulator.

3) Start the emulator. When the emulator starts, you should see the "ChatDemo" in the list of available applications.

4) Launch the ChatDemo TV app.

Once the ChatDemo is open in the emulator, you should see a green status indicator at the top right that says "Connected". Also, near the bottom you should see a 6 digit "pin code". If both of these are true, then your environment is setup correctly and ready to go.





Sample Mobile Application

Next, install either the Android or iOS sample application to test with the sample TV application. We'll use the Android example in these instructions.

1) Open the **examples/ChatDemo/mobile/android** folder in the Android API

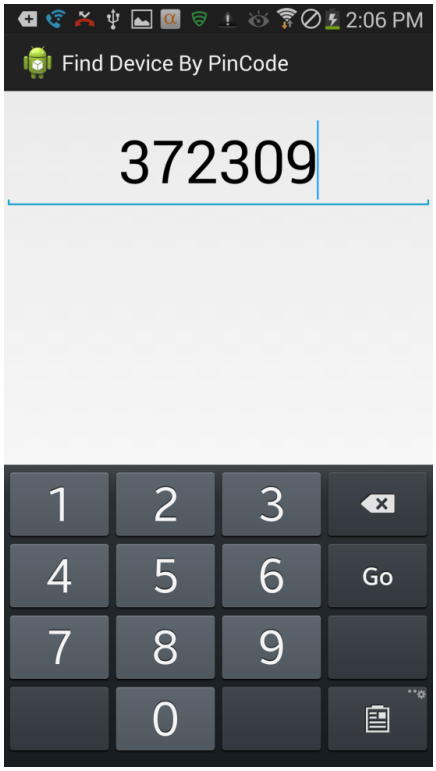
2) Install the ChatDemo-1.1.11.apk onto your Android device.
Connect an Android device to your computer via usb. Make sure your Android device is in "developer mode". Run ***adb install ChatDemo-1.1.11.apk*** *from the command line*

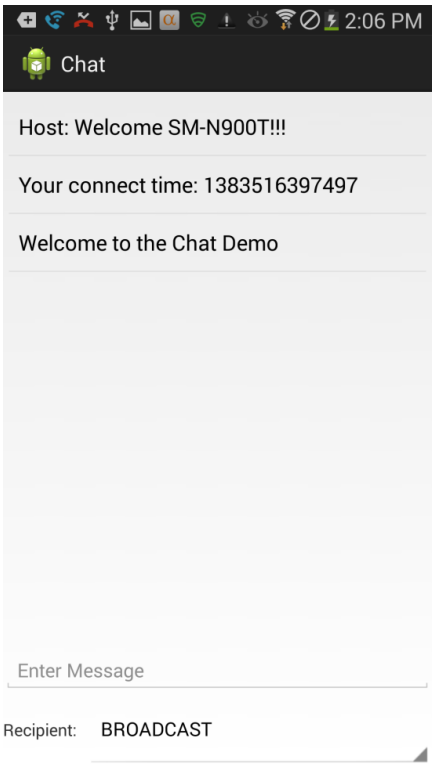
3) Launch the ChatDemo Android application.

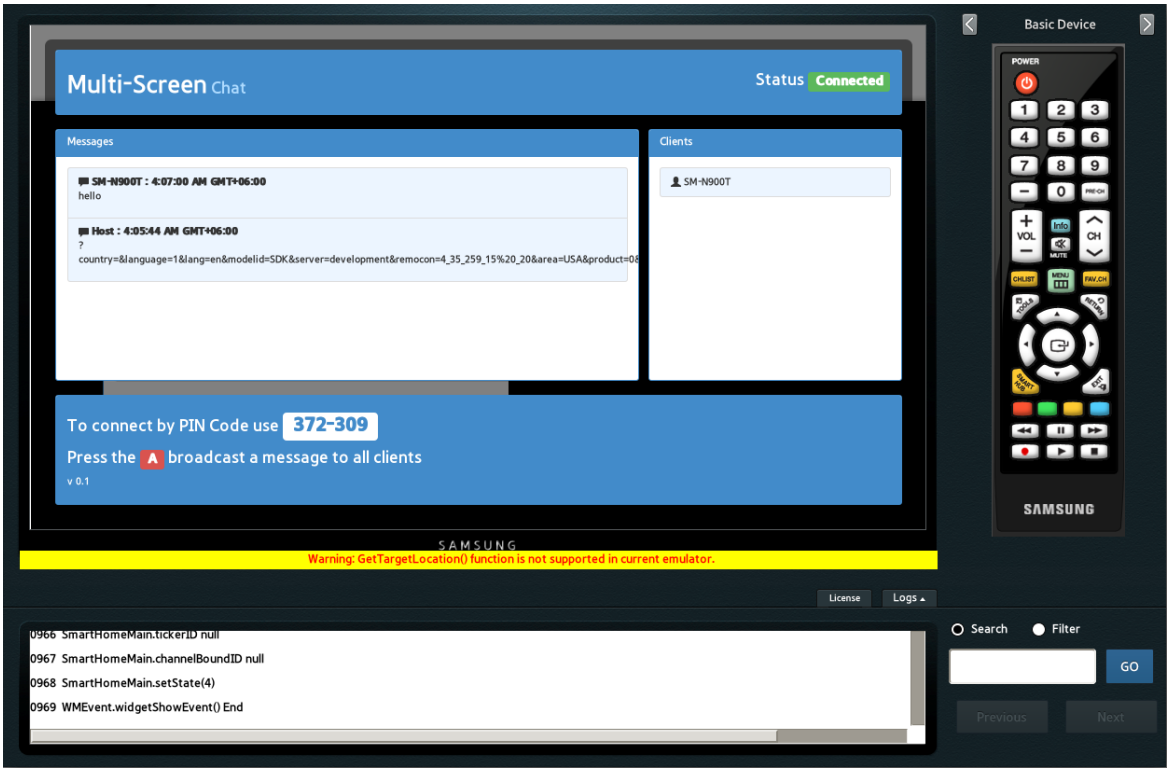
4) Select "Find Device by PinCode", enter the pin code displayed in the emulator TV application, and select "Go".
If successful, you should see the display name of your TV emulator and its IP address.

5) Select "Connect to Channel"

If everything is working correctly at this point, you should be connected to the TV application. Your Android device name will appear in the "Clients" list in the TV application, and you should receive a welcome message in the mobile app. Next, you can try sending messages from the mobile app to the TV application. You can also send messages from the TV application to the mobile application by pressing the "red" button on the virtual TV remote in the emulator.


2:06 PM
Find Device By PinCode
372309
1
2
3
4
5
6
Go
7
8
9
0


2:06 PM
Chat
Host: Welcome SM-N900T!!!
Your connect time: 1383516397497
Welcome to the Chat Demo
Enter Message
Recipient:
BROADCAST


Multi-Screen Chat
Status Connected
Messages
SM-N900T : 4:07:00 AM GMT+06:00
hello
Host : 4:05:44 AM GMT+06:00
Clients
SM-N900T
To connect by PIN Code use 372-309
Press the A broadcast a message to all clients
v 0.1
SAMSUNG
Warning: GetTargetLocation() function is not supported in current emulator.
License
Logs
0966 SmartHomeMain.tickerID null
0967 SmartHomeMain.channelBoundID null
0968 SmartHomeMain.setState(4)
0969 WMEvent.widgetShowEvent() End
Basic Device
POWER
SAMSUNG
Search
Filter
GO
Previous
Next

Basic Concepts

Before you begin developing for the Multi-Screen SDK, it's important to understand some of the concepts and terminology used in the APIs first.

The Device Class

Your first interaction with the APIs will be with the Device class. The Device class provides your mobile application with the ability to "discover" a compatible SmartTV. Once a device is discovered, you can use an instance of the Device class to launch an application on the TV, or connect to a "Channel".

Getting a reference to the "current" device in a TV application

Applications that run on the TV must get a reference to a Device instance. In this case, the TV application needs to get the "current" device.

Example Javascript code (TV Application):

```
webapis.multiscreen.Device.getCurrent(

        // success callback
        function(device) {

                console.log(device);

        },

        // error callback
        function(error) {

                console.error(error);

        }

});
```

Finding TV Devices from a mobile application

There are 2 ways to discover a TV device from a mobile application:

1) Search
   This will scan your local network and asynchronously return a list of found TV devices.

   Example Android code:

```
Device.search(new DeviceAsyncResult<List<Device>>() {

        // success callback
        public void onResult(final List devices) {

        }

        // error callback
        public void onError(final DeviceError error) {
```

```
        }

});
```

2) PinCode

You can also get a single device instance from your mobile application via a pin code. TV applications can optionally display a pin code to make it easier for mobile devices to discover and connect.

Example Android code:

```
Device.getByCode(pinCode, new DeviceAsyncResult<Device>() {

        // success callback
        public void onResult(final Device device) {

        }

        // error callback
        public void onError(DeviceError error) {

        }

});
```

The Application Class

The Application class is only available in the mobile APIs. It allows you to get a reference to your corresponding TV application, launch the application on the TV, terminate the application on the TV, and get its status.

Getting a reference to a TV Application

In order to get a reference to a TV application, you must know its “runTitle”. The “runTitle” is a special id that you set in your TV application’s config.xml.

See “Creating TV Applications” below for more information on the “runTitle”

Example Android code:

```
String runTitle = “myRunTitle”;
device.getApplication(runTitle, new AsyncResult<Application>() {

        // success callback
        public void onResult(Application application) {

        }

        // error callback
        public void onException(Exception e) {

        }
```

```
});
```

Launching a TV application from the mobile API

Once you have a reference to a TV application, you can launch it on the TV. You can also pass parameters to the TV app during the launch phase. These parameters will be available to the TV app on the URL.

Example Android code:

```
Map<String, String> parameters = new HashMap<String, String>();
parameters.put("launchedBy", "mobile);

application.launch(parameters, new AsyncResult<Boolean>) {

        // success callback
        public void onResult(Application application) {

        }

        // error callback
        public void onException(Exception e) {

        }

});
```

Getting the status of a TV Application

You can also get the latest status of a TV application, in order to determine if the TV application is running. The Application.Status enum provides the available statuses of your TV application.

Example Android code:

```
application.updateStatus(new AsyncResult<Status>) {

        // success callback
        public void onResult(Status status) {

        }

        // error callback
        public void onException(Exception e) {

  }

});
```

The Channel Class

Once you have a reference to a Device instance, you can "open" or "connect" to a Channel. A Channel is simply a communication channel that clients connect to, and provides capabilities to send and receive messages to/from any client.

Opening a Channel from a TV Application

NOTE: Only TV Applications can "open" a channel. A mobile application can only "connect" to a channel.

Example Javascript code (TV Application):

```
var channelId = "myChannelId";

var clientAttributes = {name:"TV Host"};

device.openChannel(channelId, clientAttributes,

	// success callback
	function(channel) {

		console.log("channel opened");

	},

	// error callback
	function(error) {

		console.error(error);

	}

});
```

Connecting to a Channel from a Mobile Application

Example Android code:

```
// Note: This must be the same channelId used in your TV application when "opening" a channel.
String channelId = "myChannelId";

Map<String, String> clientAttributes = new HashMap<String, String>();
clientAttributes.put("name", "Mobile Client");

device.connectToChannel(channelId, clientAttributes, new
DeviceAsyncResult<Channel>() {

	// success callback
	public void onResult(final Channel channel, final boolean reachable) {

	}

	// error callback
	public void onError(final DeviceError error) {

	}

});
```

Sending Messages via the Channel

Once clients are connected to a channel, they can freely exchange messages. Clients can send a message to a single client, a list of clients, or all clients in several ways.

Example Javascript code:

```
channel.send("Hello", clientID)                // Sends to a client by id

channel.send("Hello", [client1ID, client2ID])  // Sends to multiple clients by id

channel.send("Hello", "host")                  // Sends to the host client (the TV
client)

channel.send("Hello", "all")                   // Sends to all connected clients
including you

channel.send("Hello", "broadcast")             // Sends to all connected clients
excluding you

channel.broadcast("Hello")                     // An alias for the above example
```

Example Android code:

```
channel.broadcast(“Hello everyone except me”);     // Sends to all connected
clients excluding you

channel.sendToAll("Hello everyone including me”);  // Sends to all connected
clients including you

channel.sendToHost("Hello TV")                     // Sends to the host client
(the TV client)

ChannelClient client = channel.getClients().host();

Client.send(“Hello TV”);                           // Sends to a specific client.
```

Receiving Messages and events via the Channel

All clients can receive messages from any other client. In addition, Clients can also receive events that notify them when other clients connect or disconnect.

Example Javascript code:

```
channel.on(“message”, function(msg, client) {

        console.log(client.attributes.name + " says, "+ msg);

});

channel.on(“clientConnect”, function(client) {
```

```
        console.log(client.attributes.name + " has connected!");
});
channel.on(“clientDisconnect”, function(client) {
        console.log(client.attributes.name + " has disconnected!");
});
```

Example Android code:

```
// implement the IChannelListener interface
public class MyListenerClass implements IChannelListener {
        // Implement required methods…
        public void onClientMessage(ChannelClient client, String message) {
                LOG.info(“onClientMessage() “ + message);
        }
        public void onClientConnected(ChannelClient client) {
                LOG.info(“onClientConnected() “ + client.getId());
        }
        public void onClientDisconnected(ChannelClient client) {
                LOG.info(“onClientDisconnected() “ + client.getId();
        }
        // See javadocs for other event methods
}
// then set the listener on the channel
channel.setListener(new MyListenerClass());
```

Channel Clients

All clients that connect to a Channel are represented as a “ChannelClient”. Each ChannelClient has an id, attributes, and a method to send a message to that client.

There is one special property on each ChannelClient called “isHost”. If a ChannelClient’s isHost property is set to true, then that client is the “Host” client meaning it is the TV client. The TV

client is the one that "*Hosts*" the channel. Only the TV client can "open" a Channel. All other clients can only "connect" to a Channel.

All clients are aware of every other client connected. Each API provides a way to get a client or a list of clients, and the API keeps the client list up to date as clients connect and disconnect.

Retrieving Clients

Example **Javascript** code:

```
channel.clients.me                    // A reference to your client instance

channel.clients.host                  // A reference to the host client instance

channel.clients.get(id)               // Returns a client with the id
```

Example **Android** code:

```
ChannelClient me = channel.getClients().me();

ChannelClient host = channel.getClients().host();

ChannelClient client = channel.getClients().get(clientId);

List<ChannelClient> clientList = channel.getClients().list();
```

Creating TV Applications

TV applications are created with HTML, Javascript, and CSS. This example will demonstrate how to setup a simple TV app for use with the Multi-Screen SDK.

Refer to samsungdforum for more information about utilizing SmartTV specific features. Follow the instructions on how to create a SmartTV applications.

index.html

In your html, add the following scripts:

- **Include the Multi-Screen TV API script:**
  `<script type="text/javascript" src="$MANAGER_WIDGET/Common/webapi/1.0/multiscreenapi.js"></script>`

- **Include the SmartTV Widget API:**
  (This will be used to notify the TV that your TV application is "ready")
  `<script type="text/javascript" src="$MANAGER_WIDGET/Common/API/Widget.js"></script>`

config.xml

In your config.xml, you will need to add a special node called “runTitle” in order for your TV application to be launched from the mobile APIs.

- RunTitle:
  <runtTitle>yourRunTitle</runTitle>

The runTitle can be anything you like, but it must be unique for all applications. It’s best to use a namespace whenever possible to avoid collision with other applications.

Javascript

Add the following to your javascript initialization code. This will get a reference to the local device, open a channel, and notify the TV that your application is ready.

```
window.addEventListener(“load”, function() {

	// Get a reference to the current device (the tv)
	webapis.multiscreen.Device.getCurrent(function(device) {

		// Save a reference to the device for later

		// Open a channel that mobile devices can connect to
		var channelId = “myChannelId”;
		var clientAttributes = {name:”Alan”};

		device.openChannel(channelId, clientAttributes, function(channel) {

			// Save a reference to the channel
			// Add event listeners to channel

			// Notify the TV that your application is “ready”
			var widgetAPI = new Common.API.Widget();
			widgetAPI.sendReadyEvent();

		});

	});

});
```

Your TV application is now ready for mobile devices to connect.

Starting a Multi-Screen experience from a Mobile application

This describes the entire process for launching the experience from the mobile device. Note

that the process starts in the mobile application, and then continues in the TV application, and finally back to the mobile application

Mobile Application

1) Discover the TV device
   `Device.search(…) or Device.getByCode(pinCode, …)`

2) Get a reference to the TV application
   `device.getApplication(runTitle, ….)`

3) Launch the TV application
   `device.launch(parameters, …)`

   `……. wait for the response from the TV Application that it has been launched`

TV Application

Now the TV application is launched.

4) Add event listener for window.load

```
window.addEventListener("load", function() {

});
```

5) Get the current Device
   `webapis.multiscreen.Device.getCurrent(successCallback, errorCallback);`

6) Open a Channel
   `device.openChannel("myChannelId", {name:"Alan"}, successCallback, errorCallback);`

7) Notify Device Ready
   `widgetAPI.sendReadyEvent();`

Mobile Application

Once the TV app is launched, the mobile application can now connect to the Channel

8) Connect to the channel
   `device.connectToChannel("myChannelId", clientAttributes, …)`

Now your mobile application and TV application are connected.