
connectSDK

Feb 24, 2023

1	One SDK Eight Media Platforms	1
2	Beam Web Apps to the Big Screen	3
3	Beam Photos, Videos, Audio, and YouTube to the Big Screen	5
4	Mirror Screen and Camera Preview to the Big Screen	7
5	Promote Your TV App	9
5.1	Connect SDK Overview	9
5.2	Use Cases	10
5.3	Supported features	11
5.4	Beam Icon	15
5.5	Sample Apps	16
5.6	Testing & Debugging	16
5.7	Download Connect SDK	17
5.8	Getting Started	18
5.9	Developer Guides	22
5.10	API References	46
5.11	Getting Started	251
5.12	Developer Guides	254
5.13	API References	260
5.14	Getting Started	289
5.15	Developer Guides	293
5.16	API References	315
5.17	TV Web Apps	393
5.18	Release	397
5.19	Article	400
5.20	Terms and Conditions	403
5.21	Cookie Policy	404
5.22	Contact	407

CHAPTER 1

One SDK Eight Media Platforms

Connect SDK is an open source framework that connects your mobile apps with multiple media device platforms.

CHAPTER 2

Beam Web Apps to the Big Screen

Integrate Connect SDK into your mobile web app, and extend the viewing experience onto the big screen.

CHAPTER 3

Beam Photos, Videos, Audio, and YouTube to the Big Screen

Integrate Connect SDK into your mobile app to beam media across multiple platforms onto the big screen.

Mirror Screen and Camera Preview to the Big Screen

Integrate Connect SDK into your mobile app on Android and iOS platforms for screen mirroring and remote camera, which mirrors the screen and camera preview onto the big screen.

Promote Your TV App

Now that you created a great TV app, promote it through your mobile app using Connect SDK.

5.1 Connect SDK Overview

Connect SDK is an open source framework that connects your mobile apps with multiple TV platforms. Because most TV platforms support a variety of protocols, Connect SDK integrates and abstracts the discovery and connectivity between all supported protocols.

To discover supported platforms and protocols, Connect SDK uses SSDP to discover services such as DIAL, DLNA, UDAP, and Roku's External Control Guide (ECG). Connect SDK also supports ZeroConf to discover devices such as Chromecast and Apple TV. Even while supporting multiple discovery protocols, Connect SDK is able to generate one unified list of discovered devices from the same network.

To communicate with discovered devices, Connect SDK integrates support for protocols such as DLNA, DIAL, SSAP, ECG, AirPlay, Chromecast, UDAP, and webOS second screen protocol. Connect SDK intelligently picks which protocol to use depending on the feature being used.

For example, when connecting to a 2013 LG Smart TV, Connect SDK uses DLNA for media playback, DIAL for YouTube launching, and UDAP for system controls. On Roku, media playback and system controls are made available through ECG, and YouTube launching through DIAL. On Chromecast, media playback occurs through the Cast protocol and YouTube is launched via DIAL.

To support the aforementioned use case without Connect SDK, a developer would need to implement DIAL, ECG, Chromecast, and DLNA in their app. With Connect SDK, discovering the three devices is handled for you. Furthermore, the method calls between each protocol is abstracted. That means you can use one method call to beam a video to Roku, 3 generations of LG Smart TVs, Apple TV, and Chromecast.

5.2 Use Cases

5.2.1 Web App Beaming

Using HTML5 and other web technologies, the capabilities and opportunity are nearly limitless.

Example: Chromecast apps, which are essentially web apps, are good examples of some possibilities of integrating Connect SDK. [Click here](#) for a list of existing Chromecast apps .

Web App beaming is supported by Connect SDK v1.3 on webOS, Apple TV, and Chromecast.

5.2.2 Photo, Video & YouTube Beaming

Integrate Connect SDK into any mobile app that contains a photo, a video or YouTube video and give users the option to beam and view their content on a larger, more social display for a more engaging experience.

Example: Trulia's mobile app shows homes for sale. Instead of crowding over a screen or passing a phone around to view the homes with friends and family, the user simply beams the photos directly to the Smart TV screen allowing everyone in the room to share in the experience.

Example: The Verge app embeds product reviews, interviews and YouTube videos within their articles. With Connect SDK integrated in the app, users could beam the content onto a Smart TV or TV set top box sharing content with co-workers.

YouTube beaming is supported by Connect SDK v1.3 on webOS, LG Smart TV '13, LG Smart TV '12, Roku 3, Chromecast, Fire TV, and many DIAL supporting devices. Photo and Video beaming is supported by Connect SDK v1.3 on webOS, LG Smart TV '13, LG Smart TV '12, Roku, Apple TV, and Chromecast.

5.2.3 Screen and Camera Preview Mirroring

Integrate Connect SDK into any mobile app and let users to mirror their screen and camera preview of the mobile device on the TV for more valuable experience.

Screen Mirroring Example: The Movie Box app is a service that provides video on mobile. With Connect SDK integrated in the app, the user experience can be expanded to a larger TV screen. This allows the app users to watch a movie with their family on the large screen on their TV.

Remote Camera Example: Tom's TV doesn't have a built-in camera, so he can't make video calls with the TV. By streaming the camera to the TV with the Connect SDK, video calls can be made on the large TV without a built-in camera or USB camera.

Screen Mirroring and Remote Camera are supported on LG Smart TV '22.

5.2.4 Promote Your TV App

If you are going to invest in building a TV app, promote its availability using your mobile app. Using Connect SDK, your mobile app can detect if a specific device is on the same network and prompt the user to install your app. If the user accepts, Connect SDK launches the device's app store deep-linked to your specific app where the user can complete the download and installation.

Example: Crunchyroll, a leading Japanese Anime and Asian media video service, has a channel on Roku. By integrating Connect SDK, they could detect a Roku device on the same network and promote their channel's availability within their app.

This use case is supported by Connect SDK v1.3 on webOS, LG Smart TV '13, and Roku.

5.2.5 Control Your TV App

Own your user's experience by allowing users to control the TV app using a mobile app. Everything from keyboard input, app navigation, even logging-in can be made easier using your mobile app.

Example: Vudu could easily integrate keyboard and mouse control allowing their users to select videos and enter credit card information using Vudu mobile app. Vudu could even pass user credentials from the mobile app to the TV app eliminating the need to login on the TV if the user is already logged in on the mobile app.

5.2.6 Hybrid

Of course, developers can provide different experience depending on each platform. Some of the newer platforms like webOS and Chromecast offer newer features

5.3 Supported features

The chart below shows which APIs are available for each device.

5.3.1 Connect SDK v1.6.0

To be updated

Apps

Feature	LG Smart we-bOS '22	LG Smart we-bOS '14	Chromecast & Android TV	Apple TV	Roku	Fire TV	LG Smart TV '13	LG Smart TV '12	DIAL	Sonos Speaker	Xbox One	LG Music Flow Speaker
Beam Web App	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No
Launch My app	Yes	Yes	Yes	No	Yes	Yes	Yes. Pairing is required	Yes. Pairing is required	Yes	No	No	No
Get list of installed apps	Yes	Yes	No	No	Yes	No	Yes. Pairing is required	Yes. Pairing is required	No	No	No	No
Mobile app to TV app messaging	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No
Deeplink into app store	Yes	Yes	No	No	Yes	No	Yes	No	No	No	No	No
Beam Youtube	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes	No
Screen Mirroring	Yes	No	No	No	No	No	No	No	No	No	No	No
Remote Camera	Yes	No	No	No	No	No	No	No	No	No	No	No

Media

Feature	LG Smart webOS '14	Chromecast & Android TV	Apple TV	Roku	Fire TV	LG Smart TV '13	LG Smart TV '12	DIAL	Sonos Speaker	Xbox One	LG Music Flow Speaker
Beam video	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No
Beam audio	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Beam photo	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No
Media pause	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No
Media stop	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Get media duration	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes
Seek media	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes
Play State Subscription	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	Yes
Get Media Info	No	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	Yes
Media Info Subscription	No	Yes	No	No	No	Yes	Yes	No	Yes	Yes	Yes
SRT subtitles	No	No	No	No	No	Yes	Yes	No	No	No	No
WebVTT subtitles	Yes	Yes	No	No	Yes	No	No	No	No	No	No

System Controls

Feature	LG Smart webOS '14	Chromecast & Android TV	Apple TV	Roku	Fire TV	LG Smart TV '13	LG Smart TV '12	DIAL	Sonos Speaker	Xbox One	LG Music Flow Speaker
Show toast alert	Yes. Pairing is required	No	No	No	No	No	No	No	No	No	No
Keyboard input	Yes. Pairing is required	No	No	Yes	No	Yes. Pairing is required	Yes. Pairing is required	No	No	No	No
5-way controls	Yes. Pairing is required	No	No	Yes	No	Yes. Pairing is required	Yes. Pairing is required	No	No	No	No
Mouse controls	Yes. Pairing is required	No	No	No	No	Yes. Pairing is required	Yes. Pairing is required	No	No	No	No
Input selector	Yes	No	No	No	No	Yes. Pairing is required	Yes. Pairing is required	No	No	No	No
Power off device	Yes. Pairing is required	No	No	No	No	Yes. Pairing is required	Yes. Pairing is required	No	No	No	No

TV Controls

Feature	LG Smart webOS '14	Chromecast & Android TV	Apple TV	Roku	Fire TV	LG Smart TV '13	LG Smart TV '12	DIAL	Sonos Speaker	Xbox One	LG Music Flow Speaker
Volume up/down	Yes	Yes	No	No	No	Yes. Pairing is required	Yes. Pairing is required	No	Yes	Yes	Yes
Set volume	Yes	Yes	No	No	No	No	Yes. Pairing is required	No	Yes	Yes	Yes
Tuner channel control	Yes. Pairing is required	No	No	No	No	Yes. Pairing is required	Yes. Pairing is required	No	No	No	No
Volume Subscription	Yes	Yes	No	No	No	Yes. Pairing is required	Yes. Pairing is required	No	Yes	Yes	Yes. Pairing is required

Playlist

Feature	LG Smart webOS '14	Chromecast & Android TV	Apple TV	Roku	Fire TV	LG Smart TV '13	LG Smart TV '12	DIAL	Sonos Speaker	Xbox One	LG Music Flow Speaker
Beam Playlist	Yes	No	No	No	No	No	No	No	Yes	No	No
Play Next	Yes	No	No	No	No	No	No	No	Yes	No	No
Play Previous	Yes	No	No	No	No	No	No	No	Yes	No	No
Jump To Track	Yes	No	No	No	No	No	No	No	Yes	No	No

5.4 Beam Icon

Connect SDK is about delivering a multi-device experience across multiple platforms. Our goal from the beginning was to solve a fragmentation problem. Therefore, instead of creating another “beam” icon and expecting users to learn one more visual artifact - we recommend you use one of the many great icons already available. Google’s Cast icon is becoming widely recognized for this use case, so consider using it. Please make sure you comply with any rules set forth by the icon creator.

5.5 Sample Apps

- API Sampler
 - Android API Sampler
 - Cordova API Sampler
 - iOS API Sampler
- Media Sampler
 - Android Media Sampler
 - Cordova Media Sampler
 - iOS Media Sampler
- Web App Sampler
 - Android Web App Sampler
 - Cordova Web App Sampler
 - iOS Web App Sampler
- Screen Mirroring Sampler
 - Android Screen Mirroring Sampler
 - Android Dual Screen Sampler
 - iOS Screen Mirroring Sampler
- Remote Camera Sampler
 - Android Remote Camera Sampler
 - iOS Remote Camera Sampler

5.6 Testing & Debugging

Due to the abstracted nature of Connect SDK, it may not be necessary for you to have a suite of test devices. For many use cases, testing on one supported platform can be sufficient.

However, depending on your application and use case, it may be advisable to test each platform before you release your application. For example, while video beaming is abstracted, each platform supports different video protocols and you should make sure that your specific app's video content is playable on your desired platform.

5.6.1 webOS

The webOS TV emulator is currently available through the LG developer portal, [download here](#).

The emulator is limited in that it cannot download/install apps from LG Store. This will limit your testing on the emulator to web app & media support. Note that the emulator's network setting has to be set to "Bridged Adapter" mode for the Emulator to be discoverable.

If you have need of production hardware, the line of LG Smart TVs with webOS are currently available from major electronic retailers.

To test the Screen Mirroring or Remote Camera feature, we recommend you purchase the targeted device (webOS TV 22).

5.6.2 Chromecast

To test your application with a Chromecast device, you need to purchase a Chromecast dongle.

5.6.3 2012 and 2013 LG Smart TVs

To test your application with LG 2012 and 2013 Smart TVs, we recommend you purchase the targeted device. The emulators available [here](#) are meant to be used exclusively for first-screen TV App development.

5.6.4 Roku

In order to test your application, you should purchase a Roku device. In general, Roku devices have the same features across all models, however Roku 3 and Roku Streaming Stick have a larger app catalog, including support for YouTube videos.

5.6.5 Fire TV

To test your application with Fire TV, you should purchase a Fire TV device.

5.6.6 Apple TV

To test your application with Apple TV, you should purchase an Apple TV device.

5.7 Download Connect SDK

Connect SDK is an open source framework licensed under the [Apache License, Version 2.0](#).

5.7.1 Connect SDK v1.6.0

iOS

- Git: [Connect-SDK-iOS](#)
- Getting Started: [Setup Instructions](#) | [Discover and Connect to Device](#)

Android

- Git: [Connect-SDK-Android](#)
- Getting Started: [Setup Instructions](#) | [Discover & Connect to Device](#)

Cordova

- Git: [Connect-SDK-Cordova-Plugin](#)
- Getting Started: [Setup Instructions](#) | [Connect Your Cordova App](#)

5.8 Getting Started

5.8.1 Modularization

Structure

The **Connect SDK repositories** are adopting a modular approach with 1.4.0 release. Our aim is to provide flexibility to the developers to be able pick and choose between the various devices. Currently you can choose whether to include **Google Cast** and **Fire TV** devices or not. We plan to include more device options in the upcoming releases.

The Connect SDK is split into modules with the help of **git submodules**. There are two options:

1. The **full** project (*Connect-SDK-iOS* and *Connect-SDK-Android*) includes three submodules: core, google-cast, and firetv and thus provides the full feature set. The latter submodules are located in the modules directory.
2. The **lite** project (*Connect-SDK-iOS-Lite* and *Connect-SDK-Android-Lite*) includes the core submodule only, therefore there is no need to download any third-party dependencies.

Please refer to the figure below displaying dependencies between different modules and libraries (for iOS and Android).

Components with a light green background are external dependencies. The dashed lines show the submodule links, whereas the solid lines depict build and/or runtime dependencies.

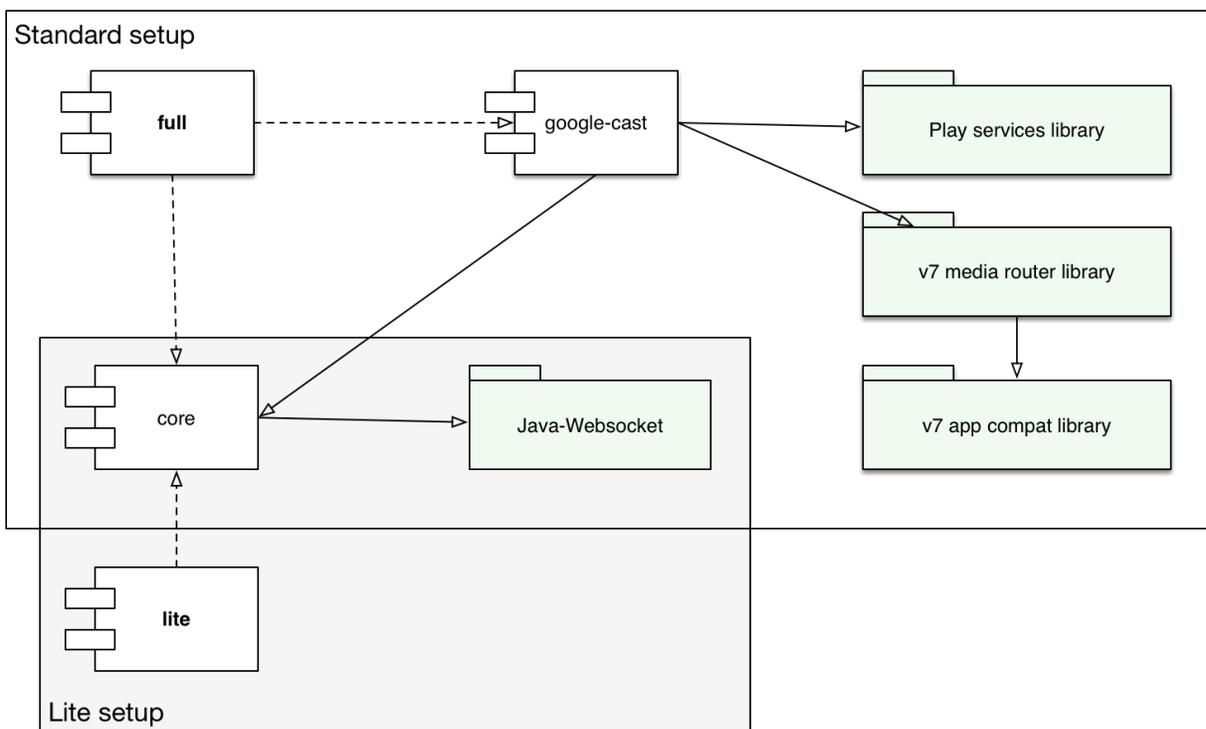


Fig. 1: Figure 1. Android SDK Component Diagram (showing Google Cast submodule as an example)

Links to the repositories are provided in the next table:

Table 1: Table 1. Links to the repositories of Android

Module	Link
full	https://github.com/ConnectSDK/Connect-SDK-Android
lite	https://github.com/ConnectSDK/Connect-SDK-Android-Lite
core	https://github.com/ConnectSDK/Connect-SDK-Android-Core
google-cast	https://github.com/ConnectSDK/Connect-SDK-Android-Google-Cast
firetv	https://github.com/ConnectSDK/Connect-SDK-Android-FireTV

Usage instructions can be found in the [full README](#) or [lite README](#).

Contributing

Since the source code is split between three repositories now (in the full version, whereas lite has only two), contributing is a bit more involved now. If you add a new feature across all the modules, you will have to create two GitHub pull requests, one for each module. Our team will check the code and merge the changes into the submodules, then update the full and lite repositories (as those just keep the project and track commits from the submodules). If you have a simpler contributing workflow in mind, please [let us know](#).

5.8.2 Setup Instructions

Dependencies

This project has the following dependencies, some of which require manual setup. If you would like to use a version of the SDK which has no manual setup, consider using the [lite version](#) of the SDK. This project can be built in Android Studio or directly with Gradle. Eclipse IDE is not supported since 1.5.0 version.

This project has the following dependencies.

- [Connect-SDK-Android-Core](#) submodule
 - Requires [Java-WebSocket](#) library
 - Requires [jmDNS](#) library
- [Connect-SDK-Android-Google-Cast](#) submodule
 - Requires [GoogleCast.framework](#)
- [Connect-SDK-Android-FireTV](#) submodule
 - Requires [Amazon Fling SDK](#)

Setup Connect SDK in Android Studio

Edit your project's `build.gradle` to add this in the “dependencies” section.

```
allprojects {
    repositories {
        google()
        jcenter()
        maven { url "https://jitpack.io" }
    }
}
```

(continues on next page)

```
//...
dependencies {
    //...
    implementation 'com.github.ConnectSDK:Connect-SDK-Android:master-SNAPSHOT'
}
```

Setup Connect SDK in Android Studio from sources

1. Open your terminal and execute these commands
 - `cd your_project_folder`
 - `git clone https://github.com/ConnectSDK/Connect-SDK-Android.git`
 - `cd Connect-SDK-Android`
 - `git submodule init`
 - `git submodule update`
2. On the root of your project directory create/modify the settings.gradle file. It should contain something like the following:

```
include ':app', ':Connect-SDK-Android'
```

3. Edit your project's build.gradle to add this in the “dependencies” section:

```
dependencies {
    //...
    implementation project(':Connect-SDK-Android')
}
```

4. Sync project with gradle files
5. Add permissions to your manifest

Permissions to include in manifest

- Required for SSDP & Chromecast/Zeroconf discovery
 - `android.permission.INTERNET`
 - `android.permission.CHANGE_WIFI_MULTICAST_STATE`
- Required for interacting with devices
 - `android.permission.ACCESS_NETWORK_STATE`
 - `android.permission.ACCESS_WIFI_STATE`
- Required for storing device pairing information
 - `android.permission.WRITE_EXTERNAL_STORAGE`
- Required for Screen Mirroring and Remote Camera
 - `android.permission.RECORD_AUDIO`
 - `android.permission.FOREGROUND_SERVICE`

```
- android.permission.CAMERA
```

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.CAMERA" />
```

Metadata for application tag

This metadata tag is necessary to enable Chromecast support.

5.8.3 Discover & Connect to Device

Initial setup

Your view controller should implement delegate/listener methods for Connect SDK's DevicePicker and ConnectableDevice classes. These methods will give you the ability to respond to device selection, ready, disconnect, and error states.

```
public class MainActivity extends Activity implements ConnectableDeviceListener {
}
```

It is helpful to retain local references to both the DiscoveryManager and the ConnectableDevice objects. In most use cases, these two classes will serve to provide most of the functionality required.

As soon as your app loads, you should instantiate the DiscoveryManager singleton and start discovery. As different devices can take a wide range of time to be discovered, it is recommended that discovery start as soon as possible after app launch.

```
private DiscoveryManager mDiscoveryManager;
private ConnectableDevice mDevice;
```

This can be initialized in the the Application class or in your Activity. You should always use getApplicationContext() since the DiscoveryManager will likely hold onto this object longer than the life of a single Activity.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    DiscoveryManager.init(getApplicationContext());

    // This step could even happen in your app's delegate
    mDiscoveryManager = DiscoveryManager.getInstance();
    mDiscoveryManager.start();
}
```

Discovery & device selection

In many cases, your user will want to select one device from a list of many. You should present the DevicePicker to the user to receive their selection. The DevicePicker includes a dynamic listing of all devices that have been discovered

on the network.

```
private void showImage() {
    DevicePicker devicePicker = new DevicePicker(this);
    AlertDialog dialog = devicePicker.getPickerDialog("Show Image", selectDevice);
    dialog.show();
}
```

Once the user has selected a device, you should immediately register for events from that device and then call the connect method.

```
AdapterView.OnItemClickListener selectDevice = new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapter, View parent, int position, long_
↪id) {
        mDevice = (ConnectableDevice) adapter.getItemAtPosition(position);
        mDevice.addListener(deviceListener);
        mDevice.connect();
    }
}
```

Capability Filtering

If your app is making use of certain device capabilities (media playback/controls, web app launching, etc), it is strongly recommended that you create filters with this information for DiscoveryManager.

Devices that are discovered & shown in the picker will be guaranteed to have the set of capabilities that you have provided. This will prevent your users from selecting a device that has not yet acquired all of its protocols.

```
CapabilityFilter videoFilter = new CapabilityFilter(
    MediaPlayer.Display_Video,
    MediaControl.Any,
    VolumeControl.Volume_Up_Down
);

CapabilityFilter imageCapabilities = new CapabilityFilter(
    MediaPlayer.Display_Image
);

DiscoveryManager.getInstance().setCapabilityFilters(videoFilter, imageCapabilities);
```

Check out the article on [capabilities](#) for more depth on this topic.

5.9 Developer Guides

5.9.1 Beam Media

A common use case with Connect SDK will be to beam a simple media file (image, video, audio) to a TV. The following is a quick example of how you can beam an image onto a TV. This example is assuming that you have discovered & connected to a device.

Beam an image file

```
String mediaURL = "http://www.connectsdk.com/files/9613/9656/8539/test_image.jpg"; //
↳ credit: Blender Foundation/CC By 3.0
String iconURL = "http://www.connectsdk.com/files/2013/9656/8845/test_image_icon.jpg";
↳ // credit: sintel-durian.deviantart.com
String title = "Sintel Character Design";
String description = "Blender Open Movie Project";
String mimeType = "image/jpeg";

MediaInfo mediaInfo = new MediaInfo.Builder(mediaURL, mimeType)
    .setTitle(title)
    .setDescription(description)
    .setIcon(iconURL)
    .build();

// These variable should be class fields
// LaunchSession mLaunchSession;
// MediaControl mMediaControl;
// ConnectableDevice mDevice;

MediaPlayer.LaunchListener listener = new MediaPlayer.LaunchListener() {
    @Override
    public void onSuccess(MediaLaunchObject object) {
        // save these object references to control media playback
        mLaunchSession = object.launchSession;
        mMediaControl = object.mediaControl;

        // you will want to enable your media control UI elements here
    }

    @Override
    public void onError(ServiceCommandError error) {
        Log.d("App Tag", "Display photo failure: " + error);
    }
};

mDevice.getMediaPlayer().displayImage(mediaInfo, listener);
```

Beam an audio/video file

```
String mediaURL = "http://www.connectsdk.com/files/8913/9657/0225/test_video.mp4"; //
↳ credit: Blender Foundation/CC By 3.0
String iconURL = "http://www.connectsdk.com/files/2013/9656/8845/test_image_icon.jpg";
↳ // credit: sintel-durian.deviantart.com
String title = "Sintel Trailer";
String description = "Blender Open Movie Project";
String mimeType = "video/mp4"; // audio/* for audio files

SubtitleInfo subtitles = null;
if (getTv().hasCapability(MediaPlayer.Subtitle_WebVTT)) {
    subtitles = new SubtitleInfo.Builder("http://ec2-54-201-108-205.us-west-2.
↳ compute.amazonaws.com/samples/media/sintel_en.vtt")
        .setMimeType("text/vtt")
        .setLanguage("en")
        .setLabel("English subtitles")
```

(continues on next page)

```
        .build();
    }
    MediaInfo mediaInfo = new MediaInfo.Builder(mediaURL, mimeType)
        .setTitle(title)
        .setDescription(description)
        .setIcon(iconURL)
        .setSubtitleInfo(subtitles)
        .build();

    // These variables should be class fields
    // LaunchSession mLaunchSession;
    // MediaControl mMediaControl;
    // ConnectableDevice mDevice;

    MediaPlayer.LaunchListener listener = new MediaPlayer.LaunchListener() {
        @Override
        public void onSuccess(MediaLaunchObject object) {
            // save these object references to control media playback
            mLaunchSession = object.launchSession;
            mMediaControl = object.mediaControl;

            // you will want to enable your media control UI elements here
        }

        @Override
        public void onError(ServiceCommandError error) {
            Log.d("App Tag", "Play media failure: " + error);
        }
    };

    mDevice.getMediaPlayer().playMedia(mediaInfo, false, listener);
```

Control media playback

In the previous example, you will notice that the success block was called with a mediaControl object. In order to control the media in the current playback session, you will need to store a reference to this mediaControl object and call control methods on that object.

```
// pause media file
mMediaControl.pause(null);

// play media file
mMediaControl.play(null);

// seek to 10 seconds
mMediaControl.seek(10000L, null);

// close media file
mMediaControl.close(null);
// or
mDevice.getMediaPlayer().closeMedia(mLaunchSession, null);
```

Beam a playlist

```
// These variables should be class fields
// LaunchSession mLaunchSession;
// MediaControl mMediaControl;
// PlaylistControl mPlaylistControl;
// ConnectableDevice mDevice;

MediaInfo mediaInfo = new MediaInfo.Builder("your-playlist.m3u", "application/x-
↳mpegurl")
    .setTitle("Playlist")
    .setDescription("Playlist description")
    .build();

mDevice.getMediaPlayer().playMedia(mediaInfo, false, new MediaPlayer.LaunchListener()
↳{
    @Override
    public void onSuccess(MediaLaunchObject object) {
        // save these object references to control media playback
        mLaunchSession = object.launchSession;
        mMediaControl = object.mediaControl;
        // playlistControl can be null if it's not supported by a service
        mPlaylistControl = object.playlistControl;
        // you will want to enable your media control UI elements here
    }

    @Override
    public void onError(ServiceCommandError error) {
        Log.d("App Tag", "Play playlist failure: " + error);
    }
});
```

Control a playlist

```
// play previous track
mPlaylistControl.previous(null);
// play next track
mPlaylistControl.next(null);
// play a track specified by index (index starts from zero)
mPlaylistControl.jumpToTrack(0, null);
```

5.9.2 Beam Web Apps

There are several platforms available which support the launching of web apps. A web app is typically run on a temporary basis in a full-screen browser instance.

Web App IDs

Both webOS and Chromecast platforms require a web app ID for API calls to launch & communicate with web apps. This web app ID is translated into your web app's URL on web app launch.

For information on creating a web app ID for webOS, please visit the [registration site](#).

To learn how to register for a Chromecast web app ID, visit [Google's app ID registration site](#).

Launch web app with identifier

Connect SDK currently supports web app launching on webOS and Chromecast devices, which both translate a web app identifier into your web app's URL.

Communicating with web apps

Bi-directional communication with your web app is made extremely simple. Data can be sent and received as strongly-typed data. For example, as a string or a keyed set of values (JSON object).

```
String webAppId = null;

// This variable should be a class field
// ConnectableDevice mDevice;

if (mDevice.getServiceByName("webOS TV") != null)
    webAppId = "5G7328DE";
else if (mDevice.getServiceByName("Chromecast") != null)
    webAppId = "3E5106AB";
else if (mDevice.getServiceByName("AirPlay") != null)
    webAppId = "http://www.example.com/";

if (webAppId == null)
    return;

mDevice.getWebAppLauncher().launchWebApp(webAppId, new WebAppSession.LaunchListener()
↳{

    @Override
    public void onError(ServiceCommandError error) {
        Log.d("App Tag", "Failed to open web app: " + error);
    }

    @Override
    public void onSuccess(WebAppSession object) {
        Log.d("App Tag", "Web app launch success");
    }
});
```

```
String webAppId = "your_web_app";
// These variables should be class fields
// WebAppSession mWebAppSession;
// WebAppSessionListener mWebAppSessionListener;
// ConnectableDevice mDevice;

mDevice.getWebAppLauncher().launchWebApp(webAppId, new WebAppSession.LaunchListener()
↳{

    @Override
    public void onError(ServiceCommandError error) {
        Log.d("App Tag", "Failed to open web app: " + error);
    }

    @Override
    public void onSuccess(WebAppSession object) {
        Log.d("App Tag", "Web app launch success");
    }
});
```

(continues on next page)

(continued from previous page)

```

mWebAppSession = object;
mWebAppSession.setWebAppSessionListener(mWebAppSessionListener);

mWebAppSession.connect(new ResponseListener() {

    @Override
    public void onError(ServiceCommandError error) {
        Log.d("App Tag", "Failed to connect to web app: " + error);
    }

    @Override
    public void onSuccess(Object object) {
        Log.d("App Tag", "Web app connect success");
    }
});
}
});
});

```

After successfully establishing a connection, you can send messages to your web app.

```
mWebAppSession.sendMessage("This is a test message", null);
```

You can also send an NSDictionary which will be received by the web app as a JSON object.

```

JSONObject message = null;
try {
    message = new JSONObject() {{
        put("someParameter", "someValue");
        put("anArray", new JSONArray() {{
            put("array value 1");
            put("array value 2");
            put("array value 3");
        }});
        put("anotherObject", new JSONObject() {{
            put("anotherParameter", "anotherValue");
        }});
    }};
} catch (JSONException e) {
    e.printStackTrace();
}

mWebAppSession.sendMessage(message, null);

```

WebAppSessionDelegate allows you to receive messages from your web app.

Beam media to web app

A common use case for web apps is the playback and control of media files. Connect SDK provides capabilities for directly playing/controlling media on a WebAppSession, provided that web app has integrated the *Connect SDK JavaScript Bridge*.

Rather than calling playMedia on your device's mediaPlayer, webAppSession provides its own mediaPlayer. After media has been beamed into the web app, the control is just like any other media session.

```

// These variable should be class fields
// LaunchSession mLaunchSession;
// MediaControl mMediaControl;
// WebAppSession mWebAppSession;

MediaPlayer.LaunchListener listener = new MediaPlayer.launchListener() {
    @Override
    public void onSuccess(MediaLaunchObject object) {
        // save these object references to control media playback
        mLaunchSession = object.launchSession;
        mMediaControl = object.mediaControl;

        // you will want to enable your media control UI elements here
    }

    @Override
    public void onError(ServiceCommandError error) {
        Log.d("App Tag", "Display photo failure: " + error);
    }
};

String mediaURL = "http://www.connectsdk.com/files/9613/9656/8539/test_image.jpg"; // ↪
↳ credit: Blender Foundation/CC By 3.0
String iconURL = "http://www.connectsdk.com/files/2013/9656/8845/test_image_icon.jpg";
↳ // credit: sintel-durian.deviantart.com
String title = "Sintel Character Design";
String description = "Blender Open Movie Project";
String mimeType = "image/jpeg";

List imageList = Arrays.asList(new ImageInfo(iconURL));
MediaInfo mediaInfo = new MediaInfo(mediaURL, mimeType, title, description, ↪
↳ imageList);

mWebAppSession.getMediaPlayer().displayImage(mediaInfo, listener);

```

5.9.3 Launch App on TV

Many TVs and streaming players include support for launching installed apps. The following is a simplified example of how to launch YouTube on a device.

Launch an app

```

// This variable should be a class field
// ConnectableDevice mDevice;
mDevice.getLauncher().launchApp("YouTube", new Launcher.AppLaunchListener() {

    @Override
    public void onError(ServiceCommandError error) {
        Log.d("App Tag", "App Launch error: " + error);
    }

    @Override
    public void onSuccess(LaunchSession object) {

```

(continues on next page)

(continued from previous page)

```

        Log.d("App Tag", "App Launch success.");
    }
});

```

Device-specific app identifiers

On each device (webOS TV, Roku, etc) apps are identified by different values. Here is an example of the different identifiers in use for the YouTube app.

- webOS: youtube.leanback.v4 (value may change with future updates)
- Netcast: 0000000000017498 (value may be different on each TV)
- DIAL: YouTube (listed in [DIAL registry](#))
- Roku: 837 (Roku-specific channel number)

Launching an app with device-specific identifiers

The following snippet shows how to detect the platform of your device and launch with the appropriate app identifier.

```

String appId = null;
// This should be a class field
// ConnectableDevice mDevice;

if (mDevice.getServiceByName(WebOSTVService.ID) != null)
    appId = "youtube.leanback.v4";
else if (mDevice.getServiceByName(NetcastTVService.ID) != null)
    appId = "0000000000017498";
else if (mDevice.getServiceByName(RokuService.ID) != null)
    appId = "837";
else if (mDevice.getServiceByName(DIALService.ID) != null)
    appId = "YouTube";

if (appId == null)
    return;

mDevice.getLauncher().launchApp(appId, new Launcher.AppLaunchListener() {

    @Override
    public void onError(ServiceCommandError error) {
        Log.d("App Tag", "App Launch error: " + error);
    }

    @Override
    public void onSuccess(LaunchSession object) {
        Log.d("App Tag", "App Launch success.");
    }

});

```

AppInfo helper object

You will notice that the previous example refers to an AppInfo object. This object is used internally by Connect SDK to manage an app's protocol-specific properties. If a device supports app list, the app list will return a set of AppInfo

objects for each app installed on the TV.

Launching an app with parameters

In most cases, a device's launcher object will allow you to pass launch parameters to your app. Connect SDK has normalized the parameter input type to a keyed set of values. These values are then parsed into the appropriate format for the protocol (XML, JSON, URL params, etc).

```
// This should be a class field
// ConnectableDevice mDevice;

JSONObject params = null;
try {
    params = new JSONObject() {{
        put("someProperty", "someValue");
    }};
} catch (JSONException e) {
    e.printStackTrace();
}

AppInfo appInfo = new AppInfo("your_app_id");
mDevice.getLauncher().launchAppWithInfo(appInfo, params, new Launcher.
↳AppLaunchListener() {

    @Override
    public void onError(ServiceCommandError error) {
        Log.d("App Tag", "App Launch error: " + error);
    }

    @Override
    public void onSuccess(LaunchSession object) {
        Log.d("App Tag", "App Launch success.");
    }
});
```

Note: Due to the variety of protocols in use, it is strongly recommended that you only use strings for the keys AND values of your parameters.

5.9.4 Discovery Manager

At the heart of Connect SDK is DiscoveryManager, a multi-protocol service discovery engine with a pluggable architecture. Much of your initial experience with Connect SDK will be with the DiscoveryManager class, as it consolidates discovered service information into ConnectableDevice objects.

DiscoveryManager supports discovering services of differing protocols by using DiscoveryProviders. Many services are discoverable over SSDP and are registered to be discovered with the SSDPDiscoveryProvider class.

As services are discovered on the network, the DiscoveryProviders will notify DiscoveryManager. DiscoveryManager is capable of attributing multiple services, if applicable, to a single ConnectableDevice instance. Thus, it is possible to have a mixed-mode ConnectableDevice object that is theoretically capable of more functionality than a single service can provide.

DiscoveryManager keeps a running list of all discovered devices and maintains a filtered list of devices that have satisfied any of your CapabilityFilters. This filtered list is used by the DevicePicker when presenting the user with a

list of devices.

Connect SDK device discovery can be started in one line.

```
DiscoveryManager.getInstance().start();
```

Features

Filtering devices by capability

It will be necessary in many cases to filter out devices that don't support a desired feature-set. DiscoveryManager provides the setCapabilityFilters method to provide for this ability.

Here is a simple example that discovers devices that support (video playback AND any media controls AND volume up/down) OR (image display).

```
CapabilityFilter videoFilter = new CapabilityFilter(
    MediaPlayer.Display_Video,
    MediaControl.Any,
    VolumeControl.Volume_Up_Down
);

CapabilityFilter imageCapabilities = new CapabilityFilter(
    MediaPlayer.Display_Image
);

DiscoveryManager.getInstance().setCapabilityFilters(videoFilter, imageCapabilities);
```

DeviceService registration

By default, Connect SDK is configured to discover all the services that it supports (webOS, Netcast, Chromecast, DIAL, & Roku). It is possible to support only a subset of these services by manually registering those services before starting DiscoveryManager for the first time.

```
DiscoveryManager.getInstance().registerDeviceService(AirPlayService.class,
↳ZeroconfDiscoveryProvider.class);
DiscoveryManager.getInstance().registerDeviceService(CastService.class,
↳CastDiscoveryProvider.class);
DiscoveryManager.getInstance().registerDeviceService(DIALService.class,
↳SSDPDiscoveryProvider.class);
DiscoveryManager.getInstance().registerDeviceService(RokuService.class,
↳SSDPDiscoveryProvider.class);
DiscoveryManager.getInstance().registerDeviceService(DLNAService.class,
↳SSDPDiscoveryProvider.class); // LG TV devices only, includes NetcastTVService
DiscoveryManager.getInstance().registerDeviceService(WebOSTVService.class,
↳SSDPDiscoveryProvider.class);
```

Pairing level

Connect SDK has support for pairing with certain devices. Having pairing disabled may reduce the number of supported capabilities that a ConnectableDevice has. Certain devices, although they may support the features you are filtering for, may not pass your CapabilityFilter if pairing is disabled.

See the *Supported Features* list for information on what devices require pairing for certain capabilities.

For the best user experience, Connect SDK has disabled pairing by default. Pairing can be enabled very easily, but it must be enabled before `DiscoveryManager` is started for the first time.

```
DiscoveryManager.getInstance().setPairingLevel(PairingLevel.ON);
```

Device store

When devices are connected to, there is certain information that is retained. This information is helpful for app re-launches, pairing, remembering commonly-used devices, and more. Connect SDK provides a `ConnectableDeviceStore` protocol to allow you to store `ConnectableDevice` information in a manner that suits your use case.

A default implementation, `DefaultConnectableDeviceStore`, will be used by `DiscoveryManager` if no other `ConnectableDeviceStore` is provided to `DiscoveryManager` when `startDiscovery` is called.

See also:

- *DiscoveryManager*
- *CapabilityFilter*
- *PairingLevel*
- *ConnectableDeviceStore*

5.9.5 Checking Capabilities

Setting up filters

When you are discovering devices you are able to specify multiple capability filters.

```
CapabilityFilter videoFilter =
    new CapabilityFilter(
        MediaPlayer.Display_Video,
        MediaControl.Any,
        VolumeControl.Volume_Up_Down);

CapabilityFilter imageFilter =
    new CapabilityFilter(
        MediaPlayer.Display_Image);

DiscoveryManager.getInstance().setCapabilityFilters(videoFilter, imageFilter);
```

Any service that is found may meet the requirements of either filter but not both. When getting the UI ready if a device might have a capability you should always check before enabling that UI component.

```
myImageButton.setEnabled(mDevice.hasCapability(MediaPlayer.Display_Image));
```

5.9.6 Resuming Apps

It may be necessary for your app to resume from a background or closed state and re-establish connection with a previously connected device. There are many ways in which Connect SDK provides information to allow for this behavior.

ConnectableDevice ID

Each `ConnectableDevice` has a unique ID assigned to it upon creation. When that device is connected to, the device store saves information about each of the device's services. The unique ID persists across app launches by attributing service UUIDs to the unique device ID in the device store.

LaunchSession

The ability to interact with an app requires some information to persist, including a session ID. This session ID may be required to close the app, as well as allow the app to accurately track certain state information.

WebAppSession

The ability to communicate with a web app requires a `LaunchSession` object and/or the web app id.

Resuming most recent connection

In order to save & reconnect to a previously connected device, all you need to keep track of is the device's ID. Assuming you are using the `ConnectableDeviceStore` included with Connect SDK, previously connected devices will persist the same ID between app launches.

When your app restarts, you should immediately start discovery and listen for device found events from `DiscoveryManager`. In these events, you can check each device's ID and call `connect` on the previously connected device.

Important note about reconnecting

Just because your device has been discovered on the network doesn't mean that all of its services/capabilities are available. You will need to set a `CapabilityFilter` on `DiscoveryManager` or manually check the `ConnectableDevice`'s capabilities before you call `connect`.

Save device ID to disk

```
ConnectableDevice device; // device you've connected to

SharedPreferences preferences = context.getSharedPreferences("MyPreferences", Context.
    ↪MODE_PRIVATE);
SharedPreferences.Editor editor = preferences.edit();

editor.putString("recentDeviceId", device.getId());
editor.commit();
```

Reconnect to device

```
ConnectableDevice mDevice;
String mRecentDeviceId;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

(continues on next page)

(continued from previous page)

```

        SharedPreferences preferences = context.getSharedPreferences("MyPreferences",
↳Context.MODE_PRIVATE);
        mRecentDeviceId = preferences.getString("recentDeviceId");

        DiscoveryManager.getInstance().setCapabilityFilters(myCapabilityFilters);
        DiscoveryManager.getInstance().addListener(this);
        DiscoveryManager.getInstance().start();
    }

    @Override
    public void onDeviceAdded(DiscoveryManager manager, ConnectableDevice device) {
        if (mRecentDeviceId != null && mDevice == null) {
            if (device.getId().equalsIgnoreCase(mRecentDeviceId)) {
                mDevice = device;
                device.addListener(this);
                device.connect();
            }
        }
    }
}

```

Resuming a web app session

Resuming a web app session is as simple as saving the WebAppSession's LaunchSession object before entering the background. It can even be serialized into a JSON object for easy cross-platform storage.

Save session info to disk

```

WebAppSession webAppSession; // retrieved from WebAppLauncher launch success block

LaunchSession launchSession = webAppSession.launchSession;
JSONObject launchSessionInfo = launchSession.toJSONObject();

SharedPreferences preferences = context.getSharedPreferences("MyPreferences", Context.
↳MODE_PRIVATE);
SharedPreferences.Editor editor = preferences.edit();

editor.putString("launchSession", launchSessionInfo.toString());
editor.commit();

```

Re-create session after device is connected/ready

```

ConnectableDevice device; // device that has been re-discovered & re-connected
WebAppSession.LaunchListener joinWebAppListener;

SharedPreferences preferences = context.getSharedPreferences("MyPreferences", Context.
↳MODE_PRIVATE);

String launchSessionData = preferences.getString("launchSession");
JSONObject launchSessionInfo = null;

```

(continues on next page)

(continued from previous page)

```
try {
    launchSessionInfo = new JSONObject(launchSessionData);
} catch (JSONException ex) {
}

if (launchSessionInfo != null) {
    LaunchSession launchSession = LaunchSession.
↳launchSessionFromJSONObject(launchSessionInfo);

    device.getWebAppLauncher().joinWebApp(launchSession, joinWebAppListener);
}
```

Low-effort re-connection option

Alternatively, you could re-join your web app with just the web app id. This could have the side effect of generating new session information for your user, which may not be desired.

```
device.getWebAppLauncher().joinWebApp("your web app id", joinWebAppListener);
```

See also:

- [Discover & Connect to Device](#)
- [Checking Capabilities](#)
- [Beam Web Apps](#)

5.9.7 Screen Mirroring

With Connect SDK integrated in the mobile app, it can cast the screen and sound into the TV screen. This allows you to extend the screen of a mobile app to a larger TV screen and share it with your family. This guide assumes that you completed the setup described in the [Setup Instructions](#).

There are two ways to display the screen to your TV.

- Screen mirroring: A way to display the entire app screen to the TV.
- Dual screen: A way to create a second screen of the app and display it on the TV while leaving the app screen separate. Dual screen is provided as a screen mirroring function.

Note: This feature is only supported on webOS TV 22.

How to Use Screen Mirroring

To use screen mirroring, follow these steps.

1. Check the Android Version

Screen mirroring runs on Android version 10 (Q, API Level 29) and higher, so you need to check the OS version when starting the app. If the OS version does not support the screen mirroring function, the function will not work or the app will close.

```

if (ScreenMirroringControl.isCompatibleOsVersion() == false) {
    // The OS version is lower than Android 10
    // and screen mirroring is not supported
}

```

2. Search Devices

Search for devices (TVs) connected to your home network. You can set the filter to only search for TVs that support the screen mirroring function. Since the search for TVs takes some time, it should be started as soon as the app is running.

```

// Initializes DiscoveryManager
DiscoveryManager.init(this);

// Sets a device search filter for devices that support screen mirroring (dual_
↳screen).
ArrayList<String> capabilities = new ArrayList<>();
capabilities.add(ScreenMirroringControl.ScreenMirroring);
CapabilityFilter filter = new CapabilityFilter(capabilities);

// Searches devices
DiscoveryManager.getInstance().setPairingLevel(DiscoveryManager.PairingLevel.ON);
DiscoveryManager.getInstance().setCapabilityFilters(filter);
DiscoveryManager.getInstance().registerDeviceService(WebOSTVService.class,
↳SSDPDiscoveryProvider.class);
DiscoveryManager.getInstance().start();

```

3. Request Permissions

The screen mirroring requires the audio permission (android.permission.RECORD_AUDIO). The permission agreement is executed only once on the first run or when there is no permission.

```

// Requests permissions
String[] permissions = new String[]{Manifest.permission.RECORD_AUDIO};
ActivityCompat.requestPermissions(this, permissions, REQUEST_CODE_ACCESS_PERMISSIONS);

// Delivers request results to onRequestPermissionsResult
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[]
↳grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);

    if (requestCode == REQUEST_CODE_ACCESS_PERMISSIONS) {
        if (hasPermission() == true) {
            // Succeeded to get permission
        } else {
            // Failed to get permission
        }
    }
}

// Checks the permissions
private boolean hasPermission() {

```

(continues on next page)

(continued from previous page)

```

    return ActivityCompat.checkSelfPermission(this, Manifest.permission.RECORD_AUDIO)
↳ == PackageManager.PERMISSION_GRANTED;
}

```

4. Get User Approval for Screen Capture

User approval is required to capture the screen. Intent data must be delivered to the screen mirroring API when consenting to screen capture.

```

// User approval is required to capture the screen
// Displays the system dialog for user approval
MediaProjectionManager projectionManager = (MediaProjectionManager)
↳ getSystemService(Context.MEDIA_PROJECTION_SERVICE);
startActivityForResult(projectionManager.createScreenCaptureIntent(), REQUEST_CODE_
↳ CAPTURE_CONSENT);

// Passes the user approval result to onActivityResult
public void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == REQUEST_CODE_CAPTURE_CONSENT) {
        if (resultCode == Activity.RESULT_OK) {
            // Succeed to get user approval
            // Intent data must be saved and delivered to screen mirroring API
            mProjectionData = data;
        } else {
            // User Approval Failed
        }
    }
}

```

5. Select a TV

Select the TV to run the screen mirroring on by using the Picker. After selecting a TV, get a ScreenMirroringControl object to use the screen mirroring API.

```

private ScreenMirroringControl mScreenMirroringControl;

AdapterView.OnItemClickListener listener = (adapter, parent, position, id) -> {
    ConnectableDevice connectableDevice = (ConnectableDevice) adapter.
↳ getItemAtPosition(position);
    mScreenMirroringControl = connectableDevice.getScreenMirroringControl();
    ...
};

// Displays a TV search picker dialog
AlertDialog dialog = new DevicePicker(this).getPickerDialog(getString(R.string.dialog_
↳ select_tv), listener);
dialog.show();

```

6. Start Screen Mirroring

Now you can run the screen mirroring. Pairing is required when you connect to a TV for the first time, and the user is informed about this.

The following runtime errors might occur while the screen mirroring is running.

- When the network connection is terminated
- When the TV is turned off
- When the screen mirroring is terminated on the TV
- When the mobile device's notification terminates the screen mirroring
- When other exceptions occurred

For these errors, it is necessary to receive the error in real-time through the listener and respond appropriately.

```
ProgressDialog progress = new ProgressDialog(this);
progress.setMessage(getString(R.string.dialog_connecting_tv));
progress.show();

// Displays the pairing pop-up
AlertDialog pairingAlert = new AlertDialog.Builder(this)
    .setTitle(getString(R.string.dialog_title_notice))
    .setCancelable(false)
    .setMessage(getString(R.string.dialog_allow_pairing))
    .setNegativeButton(android.R.string.ok, null)
    .create();

// Start the screen mirroring
// Each progress is passed through a callback function
mScreenMirroringControl.startScreenMirroring(this, mProjectionData, new
↳ScreenMirroringStartListener() {
    // When connecting to a TV for the first time, a pop-up about the mobile
↳connection is displayed on the TV,
    // and a pairing procedure is required once in which the user selects [OK] with
↳the remote control
    // To do this, the app should display a pop-up with information about pairing
    public void onPairing() {
        pairingAlert.show();
    }

    // This is a callback function when the screen mirroring starts
    // and whether or not it succeeds is passed through the result parameter
    public void onStart(boolean result, Presentation secondScreen) {
        updateButtonVisibility();
        pairingAlert.dismiss();
        progress.dismiss();

        if (result == true) Toast.makeText(ScreenMirroringActivity.this, getString(R.
↳string.toast_start_completed), Toast.LENGTH_SHORT).show();
        else Toast.makeText(ScreenMirroringActivity.this, getString(R.string.toast_
↳start_failed), Toast.LENGTH_SHORT).show();
    }
});

// This is a callback function when an unexpected error occurs while running the
↳screen mirroring
```

(continues on next page)

(continued from previous page)

```
// An error occurs when the network is disconnected, or the TV is shut down, etc.
mScreenMirroringControl.setErrorListener(this, error -> {
    // Error occurred
});
```

7. Stop Screen Mirroring

When you want to stop mirroring, call `stopScreenMirroring`.

```
// Stops screen mirroring. Whether or not to stop normally is passed through the
↳result parameter
// Abnormal termination is a case in which screen mirroring is stopped without
↳running, etc.
mScreenMirroringControl.stopScreenMirroring(this, result -> {
    Toast.makeText(ScreenMirroringActivity.this, getString(R.string.toast_stopped),
↳Toast.LENGTH_SHORT).show();
    updateButtonVisibility();
});

// Stops device search
DiscoveryManager.getInstance().stop();
DiscoveryManager.destroy();
```

How to Use Dual Screen

Dual screen is a function that creates a second screen, separate from the app screen, and displays it on the TV. The basic procedure is the same as with the screen mirroring above, and only the differences are explained below. When mirroring starts, you just need to deliver the user-defined second screen class.

Define Second Screen

Inherit Android Presentation class to define a second screen class for dual screen.

```
public class SecondScreenDemo extends Presentation implements SnakeGameListener {
    private Context mOuterContext;

    public SecondScreenDemo(@NonNull Context outerContext, @NonNull Display display) {
        super(outerContext, display);
        mOuterContext = outerContext;
    }

    @Override
    public void onCreate(@NonNull Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.snake_game_second_screen_layout);
    }
    ...
}
```

Start Dual Screen

Dual screen starts mirroring the screen by using the user-defined, Presentation inherited class. When the mobile device is connected to the TV, it creates a virtual display for the second screen, creates an instance of the second screen class, and passes it to the onStart callback. The user can then access the Second Screen class to control the dual screen.

```
mScreenMirroringControl.startScreenMirroring(this, projectionData, SecondScreenDemo.  
↪class, new ScreenMirroringControl.ScreenMirroringStartListener() {  
    ...  
  
    // This is a callback function when screen mirroring starts  
    // and whether or not it succeeds is passed through the result parameter  
    public void onStart(boolean result, Presentation secondScreen) {  
        updateButtonVisibility();  
        pairingAlert.dismiss();  
        progress.dismiss();  
  
        if (result == true) Toast.makeText(getBaseContext(), getString(R.string.toast_  
↪start_completed), Toast.LENGTH_SHORT).show();  
        else Toast.makeText(getBaseContext(), getString(R.string.toast_start_failed),  
↪Toast.LENGTH_SHORT).show();  
  
        if (secondScreen != null) {  
            mSecondScreenDemo = (SecondScreenDemo) secondScreen;  
            mSecondScreenDemo = mSecondScreenDemo.start();  
        }  
    }  
});
```

5.9.8 Remote Camera

With Connect SDK integrated in the mobile app, it can display camera preview on the TV screen. This allows you to use your mobile device's camera as a remote camera for the TV that does not have an internal or USB camera. This guide assumes that you completed the setup described in the *Setup Instructions*.

Note: This feature is only supported on webOS TV 22.

How to Use Remote Camera

To use a remote camera, follow the steps below.

1. Check the Android Version

The remote camera function is supported by Android 7 (N, API Level 24) and higher. When you run the app, check the OS version to see if the remote camera is available. If the OS version does not support the remote camera function, the function will not work or the app will close.

```
if (RemoteCameraApi.getInstance().isCompatibleOsVersion() == false) {  
    // The OS version is lower than Android 7  
    // and remote camera is not supported  
}
```

2. Search Devices

Search for devices (TVs) connected to your home network. You can set the filter to only search for TVs that support the remote camera function. Since the search for TVs takes some time, it should be started as soon as the app is running.

```
// Initializes DiscoveryManager
DiscoveryManager.init(this);

// Sets a device search filter for devices that support remote camera
ArrayList<String> capabilities = new ArrayList<>();
capabilities.add(RemoteCameraControl.RemoteCamera);
CapabilityFilter filter = new CapabilityFilter(capabilities);

// Searches devices
DiscoveryManager.getInstance().setPairingLevel(DiscoveryManager.PairingLevel.ON);
DiscoveryManager.getInstance().setCapabilityFilters(filter);
DiscoveryManager.getInstance().registerDeviceService(WebOSTVService.class,
↳SSDPDiscoveryProvider.class);
DiscoveryManager.getInstance().start();
```

3. Request Permissions

The remote camera function requires the camera permission (`android.permission.CAMERA`) and audio permission (`android.permission.RECORD_AUDIO`). The user must grant these permissions when the remote camera is first executed.

```
// Requests permissions
String[] permissions = new String[]{android.permission.CAMERA, Manifest.permission.
↳RECORD_AUDIO};
ActivityCompat.requestPermissions(this, permissions, REQUEST_CODE_ACCESS_PERMISSIONS);

// Delivers request results to onRequestPermissionsResult
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[]
↳grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);

    if (requestCode == REQUEST_CODE_ACCESS_PERMISSIONS) {
        if (hasPermission() == true) {
            // Succeeded to get permission
        } else {
            // Failed to get permission
        }
    }
}

// Checks the permissions
private boolean hasPermission() {
    return ActivityCompat.checkSelfPermission(this, Manifest.permission.CAMERA) ==
↳PackageManager.PERMISSION_GRANTED &&
        ActivityCompat.checkSelfPermission(this, Manifest.permission.RECORD_AUDIO) ==
↳PackageManager.PERMISSION_GRANTED;
}
```

4. Select a TV

Select the TV to run the remote camera on by using the Picker. After selecting a TV, get a RemoteCameraControl object to use the remote camera API.

```
private RemoteCameraControl mRemoteCameraControl ;

AdapterView.OnItemClickListener listener = (adapter, parent, position, id) -> {
    ConnectableDevice connectableDevice = (ConnectableDevice) adapter.
    ↪getItemAtPosition(position);
    mRemoteCameraControl = connectableDevice.getRemoteCameraControlControl();
    ...
};

// Displays a TV search picker dialog
AlertDialog dialog = new DevicePicker(this).getPickerDialog(getString(R.string.dialog_
    ↪select_tv), listener);
dialog.show();
```

5. Start Remote Camera

Now you can run the remote camera. First, create a SurfaceView component to show a camera preview, and then pass its Surface as a parameter. If the preview is not needed, set the Surface to null. In addition, set initial values such as the microphone mute settings or the camera lens direction and pass them as parameters. Pairing is required when you connect to a TV for the first time, and the user is informed about it.

```
// Create a SurfaceView to display the camera preview
SurfaceView surfaceView = findViewById(R.id.surfaceView);
SurfaceHolder holder = surfaceView.getHolder();

holder.addCallback(new SurfaceHolder.Callback() {
    public void surfaceCreated(SurfaceHolder holder) {
        // When the SurfaceView is created, pass it as an argument to request the_
    ↪remote camera to start
        startRemoteCamera(holder.getSurface());
    }
    ...
});

private void startRemoteCamera(Surface surface) {
    AlertDialog pairingAlert = new AlertDialog.Builder(this)
        .setTitle(getString(R.string.dialog_title_notice))
        .setCancelable(false)
        .setMessage(getString(R.string.dialog_allow_pairing))
        .setNegativeButton(android.R.string.ok, null)
        .create();

    // Starts the remote camera
    // Each progress is passed through a callback function
    mRemoteCameraControl.startRemoteCamera(this, surface, mMicMute, mLensFacing, new_
    ↪RemoteCameraStartListener() {
        // When connecting to a TV for the first time, a pop-up about the mobile_
    ↪connection is displayed on the TV,
        // and a pairing procedure procedure is required once in which the user_
    ↪selects [OK] with the remote control.
```

(continues on next page)

(continued from previous page)

```

// To do this, the app should display a pop-up with information about pairing
public void onPairing() {
    pairingAlert.show();
}

// This is a callback function when the remote camera starts
// and whether or not it succeeds is passed through the result parameter
public void onStart(boolean result) {
    if (result == true) {
        mPlayingAlert.show();
    } else {
        Toast.makeText(CameraPreviewActivity.this, getString(R.string.toast_
↪start_failed), Toast.LENGTH_SHORT).show();
        finish();
    }
    pairingAlert.dismiss();
}
});

// Handles the callback when camera properties are changed on the TV
mRemoteCameraControl.setPropertyChangeListener(this, property -> {
    Toast.makeText(this, getString(R.string.toast_property_changed) + ": " +
↪property, Toast.LENGTH_SHORT).show();
});

// This is a callback function when an unexpected error occurs while running the
↪remote camera
// An error occurs when the network is disconnected, the TV is shut down, etc.
mRemoteCameraControl.setErrorListener(this, error -> {
    Toast.makeText(this, getString(R.string.toast_running_error) + ": " + error,
↪Toast.LENGTH_SHORT).show();
    mPlayingAlert.dismiss();
});
}

```

6. Start Camera Playback

You can designate `setCameraPlayingListener` to receive a callback when camera stream transmission and playback start by selecting the mobile device's camera on the TV. When the camera playback starts on the TV, take appropriate actions such as removing pop-ups.

```

// Handles the callback function when the remote camera preview screen starts by
↪selecting the mobile on the TV
mRemoteCameraControl.setCameraPlayingListener(this, () -> {
    Toast.makeText(this, getString(R.string.toast_play_started), Toast.LENGTH_SHORT).
↪show();
    mPlayingAlert.dismiss();
});

```

7. Stop Remote Camera

When you want to stop the remote camera, call `stopRemoteCamera`.

```
mRemoteCameraControl.stopRemoteCamera(this, result->{  
    ...  
});
```

Features

Change Camera Property

You can change camera properties such as brightness and AWB on the TV, and you can receive callbacks by designating a `setPropertyChangeListener` listener.

```
// Handles the callback function when changing camera properties on the TV  
mRemoteCameraControl.setPropertyChangeListener(this, property -> {  
    Toast.makeText(this, getString(R.string.toast_property_changed) + ": " + property,  
    ↪ Toast.LENGTH_SHORT).show();  
});
```

Handle Runtime Errors

The following runtime error might occur while the remote camera is running.

- When the network connection is terminated
- When the TV is turned off
- When the remote camera is terminated on the TV
- When the mobile device's notification terminates the remote camera
- When other exceptions occurred

For these errors, it is necessary to receive the error in real-time through the listener and respond appropriately.

```
// This is a callback function when an unexpected error occurs while running a remote,  
↪ camera  
// An error occurs when the network connection is disconnected, the TV is shut down,  
↪ etc.  
mRemoteCameraControl.setErrorListener(this, error -> {  
    Toast.makeText(this, getString(R.string.toast_running_error) + ": " + error,  
    ↪ Toast.LENGTH_SHORT).show();  
    mPlayingAlert.dismiss();  
});
```

Set the Microphone Mute State

If you change the microphone mute state, it must be transmitted. The app must maintain the current mute setting value.

```
mRemoteCameraControl.setMicMute(this, mMicMute); // true or false
```

Switch between Front and Back Cameras

When the direction of the camera is switched between front and rear, the camera direction is transmitted. The app must maintain the current camera direction value.

```
mRemoteCameraControl.setLensFacing(this, mLensFacing); // RemoteCameraApi.LENS_FACING_
↳BACK or RemoteCameraApi.LENS_FACING_FRONT
```

5.9.9 FAQ

When do I start the DiscoveryManager?

We recommend starting the DiscoveryManager when the app is started so that devices can be discovered and ready for use by the time the UI is loaded.

If you need to start the discovery later or only during a specific activity within your app you should be aware that it can take a few seconds for devices to be discovered.

How do I reconnect to a device on resume?

When your app goes into the background you can hold onto a ConnectableDevice object. When your app resumes you have the reference to the ConnectableDevice and you can listen for the Device ready function. Once the device is ready you can call connect and begin using it again.

How do I re-connect to a Web App when app resumes?

When a WebApp is launched on a TV you get a reference to that WebApp's WebAppSession object. When your phone's application goes into the background you can hold onto that WebAppSession object for the next time your application is in the foreground. Once your app is in the foreground again and you get a ConnectableDevice object.

```
public void onDeviceReady(ConnectableDevice device);
```

Then once the method is called you can use the stored WebAppSession object to continue to send commands to the running app.

How do I get the number of devices discovered?

When you start an app you should always assume that there are 0 devices discovered. Using the DiscoveryManagerDelegate you will be notified whenever a new device is discovered and an existing device has been lost.

```
public void onDeviceAdded(DiscoveryManager manager, ConnectableDevice device);
public void onDeviceRemoved(DiscoveryManager manager, ConnectableDevice device);
```

When either of these methods are called you can reference the compatibleDevices property of the sharedManager to get a complete list of devices that match your filters.

When there are no compatible devices your UI should reflect this by hiding the beam icon.

How do I get an ADHoc list of devices?

When you specify your device filters you may have devices that don't support every feature. If you are searching for all devices that can either display an image or play a YouTube video then you want to show a list of all the devices that can show an image.

To do this you will need to check that each device in the compatibleDevices array has the capabilities that you are looking for.

```
public List getImageDevices() {
    List imageDevices = new ArrayList();

    for (ConnectableDevice device : DiscoveryManager.getInstance().
    ↪getCompatibleDevices().values()) {
        if (device.hasCapability(MediaPlayer.Display_Image))
            imageDevices.add(device);
    }

    return imageDevices;
}
```

How do I show an image or video from my device?

All videos that are sent with the Connect SDK are links to external web content and your device is no different. You can setup a quick HTTP server and pass the url of your phone with connect SDK. The media player will reach to your HTTP server and stream your content right from there.

There are some pre-made libraries that already do the heavy lifting for you.

Checkout: [NanoHttpd](#)

5.10 API References

5.10.1 Discovery

CapabilityFilter

`com.connectsdk.discovery.CapabilityFilter`

CapabilityFilter is an object that wraps a List of required capabilities. This CapabilityFilter is used for determining which devices will appear in DiscoveryManager's compatibleDevices array. The contents of a CapabilityFilter's array must be any of the string constants defined in the Capability Class constants.

CapabilityFilter values

Here are some examples of values for the Capability constants.

- `MediaPlayer.Display_Video` = "MediaPlayer.Display.Video"
- `MediaPlayer.Display_Image` = "MediaPlayer.Display.Image"
- `VolumeControl.Volume_Subscribe` = "VolumeControl.Subscribe"
- `MediaControl.Any` = "MediaControl.Any"

All Capability header files also define a constant array of all capabilities defined in that header (ex. `kVolumeControl-Capabilities`).

AND/OR Filtering

CapabilityFilter is an AND filter. A ConnectableDevice would need to satisfy all conditions of a CapabilityFilter to pass.

The `DiscoveryManager` `capabilityFilters` is an OR filter. a `ConnectableDevice` only needs to satisfy one condition (`CapabilityFilter`) to pass.

Examples

Filter for all devices that support video playback AND any media controls AND volume up/down.

```
List<String> capabilities = new ArrayList<String>();
capabilities.add(MediaPlayer.Display_Video);
capabilities.add(MediaControl.Any);
capabilities.add(VolumeControl.Volume_Up_Down);

CapabilityFilter filter =
    new CapabilityFilter(capabilities);
DiscoveryManager.getInstance().setCapabilityFilters(filter);
```

Filter for all devices that support (video playback AND any media controls AND volume up/down) OR (image display).

```
CapabilityFilter videoFilter =
    new CapabilityFilter(
        MediaPlayer.Display_Video,
        MediaControl.Any,
        VolumeControl.Volume_Up_Down);

CapabilityFilter imageFilter =
    new CapabilityFilter(
        MediaPlayer.Display_Image);

DiscoveryManager.getInstance().setCapabilityFilters(videoFilter, imageFilter);
```

Properties

`List<String> capabilities = new ArrayList<String>()`

List of capabilities required by this filter. This property is readonly use the `addCapability` or `addCapabilities` to build this object.

Methods

`CapabilityFilter ()`

Create an empty `CapabilityFilter`.

`CapabilityFilter (String... capabilities)`

Create a `CapabilityFilter` with the given array of required capabilities.

Parameters:

- `capabilities` – Capabilities to be added to the new filter

`CapabilityFilter (List<String> capabilities)`

Create a `CapabilityFilter` with the given array of required capabilities.

Parameters:

- capabilities – List of capability names (see capability class files for String constants)

void **addCapability** (String *capability*)

Add a required capability to the filter.

Parameters:

- capability – Capability name to add (see capability class files for String constants)

void **addCapabilities** (List<String> *capabilities*)

Add array of required capabilities to the filter. (see capability class files for String constants)

Parameters:

- capabilities – List of capability names

void **addCapabilities** (String... *capabilities*)

Add array of required capabilities to the filter. (see capability classes files for String constants)

Parameters:

- capabilities – String[] of capability names

DevicePicker

`com.connectsdk.device.DevicePicker`

Overview

The DevicePicker is provided by the DiscoveryManager as a simple way for you to present a list of available devices to your users.

In Depth

By calling the `getPickerDialog` you will get a reference to the AlertDialog that will be updated automatically updated as compatible devices are discovered.

Methods

DevicePicker (Activity *activity*) Creates a new DevicePicker

Parameters:

- activity – Activity that DevicePicker will appear in

ListView **getListView** ()

void **pickDevice** (*ConnectableDevice device*) Sets a selected device.

Parameters:

- device – Device that has been selected.

void **cancelPicker** () Cancels pairing with the currently selected device.

AlertDialog getPickerDialog (String *message*, final OnItemClickListener *listener*) This method will return an AlertDialog that contains a ListView with an item for each discovered ConnectableDevice.

Parameters:

- *message* – The title for the AlertDialog
- *listener* – The listener for the ListView to get the item that was clicked on

DiscoveryManager

`com.connectsdk.discovery.DiscoveryManager`

Overview

At the heart of Connect SDK is DiscoveryManager, a multi-protocol service discovery engine with a pluggable architecture. Much of your initial experience with Connect SDK will be with the DiscoveryManager class, as it consolidates discovered service information into ConnectableDevice objects.

In depth

DiscoveryManager supports discovering services of differing protocols by using DiscoveryProviders. Many services are discoverable over [SSDP](#) and are registered to be discovered with the SSDPDiscoveryProvider class.

As services are discovered on the network, the DiscoveryProviders will notify DiscoveryManager. DiscoveryManager is capable of attributing multiple services, if applicable, to a single ConnectableDevice instance. Thus, it is possible to have a mixed-mode ConnectableDevice object that is theoretically capable of more functionality than a single service can provide.

DiscoveryManager keeps a running list of all discovered devices and maintains a filtered list of devices that have satisfied any of your CapabilityFilters. This filtered list is used by the DevicePicker when presenting the user with a list of devices.

Only one instance of the DiscoveryManager should be in memory at a time. To assist with this, DiscoveryManager has static method at sharedManager.

Example:

```
DiscoveryManager.init(getApplicationContext());
DiscoveryManager discoveryManager = DiscoveryManager.getInstance();
discoveryManager.addListener(this);
discoveryManager.start();
```

Inner Classes

- *PairingLevel*

Methods

static void init (Context *context*) Initializes the Discovery manager with a valid context. This should be done as soon as possible and it should use `getApplicationContext()` as the Discovery manager could persist longer than the current Activity.

```
DiscoveryManager.init(getApplicationContext());
```

Parameters:

- context

static void **destroy** ()

static void init (**Context** *context*, **ConnectableDeviceStore** *connectableDeviceStore*) Initializes the Discovery manager with a valid context. This should be done as soon as possible and it should use `getApplicationContext()` as the Discovery manager could persist longer than the current Activity.

This accepts a `ConnectableDeviceStore` to use instead of the default device store.

```
MyConnectableDeviceStore myDeviceStore = new MyConnectableDeviceStore();
DiscoveryManager.init(getApplicationContext(), myDeviceStore);
```

Parameters:

- context
- connectableDeviceStore

static *DiscoveryManager* getInstance () Get a shared instance of `DiscoveryManager`.

void addListener (***DiscoveryManagerListener*** *listener*) Listener which should receive discovery updates. It is not necessary to set this listener property unless you are implementing your own device picker. Connect SDK provides a default `DevicePicker` which acts as a `DiscoveryManagerListener`, and should work for most cases.

If you have provided a `capabilityFilters` array, the listener will only receive update messages for `ConnectableDevices` which satisfy at least one of the `CapabilityFilters`. If no `capabilityFilters` array is provided, the listener will receive update messages for all `ConnectableDevice` objects that are discovered.

Parameters:

- listener – (optional) `DiscoveryManagerListener` with methods to be called on success or failure

void removeListener (***DiscoveryManagerListener*** *listener*) Removes a previously added listener

Parameters:

- listener – (optional) `DiscoveryManagerListener` with methods to be called on success or failure

void setCapabilityFilters (***CapabilityFilter***... *capabilityFilters*) **Parameters:**

- `capabilityFilters`

void setCapabilityFilters (**List**<***CapabilityFilter***> *capabilityFilters*) **Parameters:**

- `capabilityFilters`

List<***CapabilityFilter***> **getCapabilityFilters** () Returns the list of capability filters.

boolean deviceIsCompatible (***ConnectableDevice*** *device*) **Parameters:**

- `device`

void start () Start scanning for devices on the local network.

void stop () Stop scanning for devices.

void setConnectableDeviceStore (***ConnectableDeviceStore*** *connectableDeviceStore*) `ConnectableDeviceStore` object which loads & stores references to all discovered devices. Pairing codes/keys, SSL certificates, recent access times, etc are kept in the device store.

ConnectableDeviceStore is a protocol which may be implemented as needed. A default implementation, DefaultConnectableDeviceStore, exists for convenience and will be used if no other device store is provided.

In order to satisfy user privacy concerns, you should provide a UI element in your app which exposes the ConnectableDeviceStore removeAll method.

To disable the ConnectableDeviceStore capabilities of Connect SDK, set this value to nil. This may be done at the time of instantiation with `DiscoveryManager.init(context, null);`.

Parameters:

- connectableDeviceStore

ConnectableDeviceStore getConnectableDeviceStore () ConnectableDeviceStore object which loads & stores references to all discovered devices. Pairing codes/keys, SSL certificates, recent access times, etc are kept in the device store.

ConnectableDeviceStore is a protocol which may be implemented as needed. A default implementation, DefaultConnectableDeviceStore, exists for convenience and will be used if no other device store is provided.

In order to satisfy user privacy concerns, you should provide a UI element in your app which exposes the ConnectableDeviceStore removeAll method.

To disable the ConnectableDeviceStore capabilities of Connect SDK, set this value to nil. This may be done at the time of instantiation with `DiscoveryManager.init(context, null);`.

Map<String, ConnectableDevice> getAllDevices () List of all devices discovered by DiscoveryManager. Each ConnectableDevice object is keyed against its current IP address.

Map<String, ConnectableDevice> getCompatibleDevices () Filtered list of discovered ConnectableDevices, limited to devices that match at least one of the CapabilityFilters in the capabilityFilters array. Each ConnectableDevice object is keyed against its current IP address.

PairingLevel getPairingLevel () The pairingLevel property determines whether capabilities that require pairing (such as entering a PIN) will be available.

If pairingLevel is set to ConnectableDevicePairingLevelOn, ConnectableDevices that require pairing will prompt the user to pair when connecting to the ConnectableDevice.

If pairingLevel is set to ConnectableDevicePairingLevelOff (the default), connecting to the device will avoid requiring pairing if possible but some capabilities may not be available.

void setPairingLevel (PairingLevel pairingLevel) The pairingLevel property determines whether capabilities that require pairing (such as entering a PIN) will be available.

If pairingLevel is set to ConnectableDevicePairingLevelOn, ConnectableDevices that require pairing will prompt the user to pair when connecting to the ConnectableDevice.

If pairingLevel is set to ConnectableDevicePairingLevelOff (the default), connecting to the device will avoid requiring pairing if possible but some capabilities may not be available.

Parameters:

- pairingLevel

Inherited Methods

void onDeviceReady (ConnectableDevice device) A ConnectableDevice sends out a ready message when all of its connectable DeviceServices have been connected and are ready to receive commands.

Parameters:

- device – ConnectableDevice that is ready for commands.

void onDeviceDisconnected (*ConnectableDevice device*) When all of a ConnectableDevice's DeviceServices have become disconnected, the disconnected message is sent.

Parameters:

- device – ConnectableDevice that has been disconnected.

void onPairingRequired (*ConnectableDevice device, DeviceService service, PairingType pairingType*) DeviceService listener proxy method.

This method is called when a DeviceService tries to connect and finds out that it requires pairing information from the user.

Parameters:

- device – ConnectableDevice containing the DeviceService
- service – DeviceService that requires pairing
- pairingType – DeviceServicePairingType that the DeviceService requires

void onCapabilityUpdated (*ConnectableDevice device, List<String> added, List<String> removed*) When a ConnectableDevice finds & loses DeviceServices, that ConnectableDevice will experience a change in its collective capabilities list. When such a change occurs, this message will be sent with arrays of capabilities that were added & removed.

This message will allow you to decide when to stop/start interacting with a ConnectableDevice, based off of its supported capabilities.

Parameters:

- device – ConnectableDevice that has experienced a change in capabilities
- added – List<String> of capabilities that are new to the ConnectableDevice
- removed – List<String> of capabilities that the ConnectableDevice has lost

void onConnectionFailed (*ConnectableDevice device, ServiceCommandError error*) This method is called when the connection to the ConnectableDevice has failed.

Parameters:

- device – ConnectableDevice that has failed to connect
- error – ServiceCommandError with a description of the failure

void onServiceAdded (*DiscoveryProvider provider, ServiceDescription serviceDescription*) This method is called when the DiscoveryProvider discovers a service that matches one of its DeviceService filters. The ServiceDescription is created and passed to the listener (which should be the DiscoveryManager). The ServiceDescription is used to create a DeviceService, which is then attached to a ConnectableDevice object.

Parameters:

- provider – DiscoveryProvider that found the service
- serviceDescription

void onServiceRemoved (*DiscoveryProvider provider, ServiceDescription serviceDescription*) This method is called when the DiscoveryProvider's internal mechanism loses reference to a service that matches one of its DeviceService filters.

Parameters:

- provider – DiscoveryProvider that lost the service
- serviceDescription

void onServiceDiscoveryFailed (DiscoveryProvider provider, ServiceCommandError error) This method is called on any error/failure within the DiscoveryProvider.

Parameters:

- provider – DiscoveryProvider that failed
- error – ServiceCommandError providing a information about the failure

void onServiceConfigUpdate (ServiceConfig serviceConfig) Parameters:

- serviceConfig

DiscoveryManagerListener

`com.connectsdk.discovery.DiscoveryManagerListener`

Overview

The DiscoveryManagerListener will receive events on the addition/removal/update of ConnectableDevice objects.

In Depth

It is important to note that, unless you are implementing your own device picker, this listener is not needed in your code. Connect SDK's DevicePicker internally acts a separate listener to the DiscoveryManager and handles all of the same method calls.

Methods

void onDeviceAdded (DiscoveryManager manager, ConnectableDevice device) This method will be fired upon the first discovery of one of a ConnectableDevice's DeviceServices.

Parameters:

- manager – DiscoveryManager that found device
- device – ConnectableDevice that was found

void onDeviceUpdated (DiscoveryManager manager, ConnectableDevice device) This method is called when a ConnectableDevice gains or loses a DeviceService in discovery.

Parameters:

- manager – DiscoveryManager that updated device
- device – ConnectableDevice that was updated

void onDeviceRemoved (DiscoveryManager manager, ConnectableDevice device) This method is called when connections to all of a ConnectableDevice's DeviceServices are lost. This will usually happen when a device is powered off or loses internet connectivity.

Parameters:

- manager – DiscoveryManager that lost device
- device – ConnectableDevice that was lost

void onDiscoveryFailed (*DiscoveryManager manager*, *ServiceCommandError error*) In the event of an error in the discovery phase, this method will be called.

Parameters:

- manager – DiscoveryManager that experienced the error
- error – NSError with a description of the failure

PairingLevel

`com.connectsdk.discovery.DiscoveryManager.PairingLevel`

Describes a pairing level for a DeviceService. It's used by a DiscoveryManager and all services.

Properties

OFF Specifies that pairing is off. DeviceService will never try to pair with a first screen device.

ON Specifies that pairing is on. DeviceService will try to pair if it is required by a first screen device.

PairingType

`com.connectsdk.service.DeviceService.PairingType`

Enumerates available pairing types. It is used by a DeviceService for implementing pairing strategy.

Properties

NONE DeviceService doesn't require pairing

FIRST_SCREEN In this mode user must confirm pairing on the first screen device (e.g. an alert on a TV)

PIN_CODE In this mode user must enter a pin code from a mobile device and send it to the first screen device

MIXED In this mode user can either enter a pin code from a mobile device or confirm pairing on the TV

5.10.2 Device

ConnectableDevice

`com.connectsdk.device.ConnectableDevice`

Overview

ConnectableDevice serves as a normalization layer between your app and each of the device's services. It consolidates a lot of key data about the physical device and provides access to underlying functionality.

In Depth

ConnectableDevice consolidates some key information about the physical device, including model name, friendly name, ip address, connected DeviceService names, etc. In some cases, it is not possible to accurately select which DeviceService has the best friendly name, model name, etc. In these cases, the values of these properties are dependent upon the order of DeviceService discovery.

To be informed of any ready/pairing/disconnect messages from each of the DeviceService, you must set a listener.

ConnectableDevice exposes capabilities that exist in the underlying DeviceServices such as TV Control, Media Player, Media Control, Volume Control, etc. These capabilities, when accessed through the ConnectableDevice, will be automatically chosen from the most suitable DeviceService by using that DeviceService's CapabilityPriorityLevel.

Methods

void setPairingType (*PairingType pairingType*) set desirable pairing type for all services

Parameters:

- pairingType

void addService (*DeviceService service*) Adds a DeviceService to the ConnectableDevice instance. Only one instance of each DeviceService type (webOS, Netcast, etc) may be attached to a single ConnectableDevice instance. If a device contains your service type already, your service will not be added.

Parameters:

- service – DeviceService to be added

void removeService (*DeviceService service*) Removes a DeviceService from the ConnectableDevice instance.

Parameters:

- service – DeviceService to be removed

void removeServiceWithId (*String serviceId*) Removes a DeviceService from the ConnectableDevice instance.

Parameters:

- serviceId – ID of the DeviceService to be removed (DLNA, webOS TV, etc)

Collection<DeviceService> getServices () Array of all currently discovered DeviceServices this ConnectableDevice has associated with it.

DeviceService getServiceByName (*String serviceName*) Obtains a service from the ConnectableDevice with the provided serviceName

Parameters:

- serviceName – Service ID of the targeted DeviceService (webOS, Netcast, DLNA, etc)

Returns: DeviceService with the specified serviceName or nil, if none exists

void removeServiceByName (*String serviceName*) Removes a DeviceService form the ConnectableDevice instance. serviceName is used as the identifier because only one instance of each DeviceService type may be attached to a single ConnectableDevice instance.

Parameters:

- serviceName – Name of the DeviceService to be removed from the ConnectableDevice.

DeviceService getServiceWithUUID (*String serviceUUID*) Returns a DeviceService from the ConnectableDevice instance. serviceUUID is used as the identifier because only one instance of each DeviceService type may be attached to a single ConnectableDevice instance.

Parameters:

- serviceUUID – UUID of the DeviceService to be returned

void addListener (*ConnectableDeviceListener listener*) Adds the ConnectableDeviceListener to the list of listeners for this ConnectableDevice to receive certain events.

Parameters:

- listener – ConnectableDeviceListener to listen to device events (connect, disconnect, ready, etc)

void setListener (*ConnectableDeviceListener listener*) Clears the array of listeners and adds the provided listener to the array. If listener is null, the array will be empty.

This method is deprecated. Since version 1.2.1, use `ConnectableDevice::addListener(ConnectableDeviceListener listener)` instead

Parameters:

- listener – ConnectableDeviceListener to listen to device events (connect, disconnect, ready, etc)

void removeListener (*ConnectableDeviceListener listener*) Removes a previously added ConenctableDeviceListener from the list of listeners for this ConnectableDevice.

Parameters:

- listener – ConnectableDeviceListener to be removed

List<*ConnectableDeviceListener*> **getListeners** ()

void connect () Enumerates through all DeviceServices and attempts to connect to each of them. When all of a ConnectableDevice's DeviceServices are ready to receive commands, the ConnectableDevice will send a onDeviceReady message to its listener.

It is always necessary to call connect on a ConnectableDevice, even if it contains no connectable DeviceServices.

void disconnect () Enumerates through all DeviceServices and attempts to disconnect from each of them.

boolean isConnectable () Whether the device has any DeviceServices that require an active connection (websocket, HTTP registration, etc)

void sendPairingKey (*String pairingKey*) Sends a pairing key to all discovered device services.

Parameters:

- pairingKey – Pairing key to send to services.

void cancelPairing () Explicitly cancels pairing on all services that require pairing. In some services, this will hide a prompt that is displaying on the device.

List<*String*> **getCapabilities** () A combined list of all capabilities that are supported among the detected DeviceServices.

boolean hasCapability (*String capability*) Test to see if the capabilities array contains a given capability. See the individual Capability classes for acceptable capability values.

It is possible to append a wildcard search term `.Any` to the end of the search term. This method will return true for capabilities that match the term up to the wildcard.

Example: `Launcher.App.Any`

Parameters:

- capability – Capability to test against

boolean hasAnyCapability (String... capabilities) Test to see if the capabilities array contains at least one capability in a given set of capabilities. See the individual Capability classes for acceptable capability values.

See hasCapability: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – Array of capabilities to test against

boolean hasCapabilities (List<String> capabilities) Test to see if the capabilities array contains a given set of capabilities. See the individual Capability classes for acceptable capability values.

See hasCapability: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – Array of capabilities to test against

boolean hasCapabilities (String... capabilities) Test to see if the capabilities array contains a given set of capabilities. See the individual Capability classes for acceptable capability values.

See hasCapability: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – Array of capabilities to test against

Launcher getLauncher () Accessor for highest priority Launcher object This method is deprecated. Use `ConnectableDevice::getCapability(Class<T> controllerClass)` method instead

MediaPlayer getMediaPlayer () Accessor for highest priority MediaPlayer object This method is deprecated. Use `ConnectableDevice::getCapability(Class<T> controllerClass)` method instead

MediaControl getMediaControl () Accessor for highest priority MediaControl object This method is deprecated. Use `ConnectableDevice::getCapability(Class<T> controllerClass)` method instead

PlaylistControl getPlaylistControl () Accessor for highest priority PlaylistControl object This method is deprecated. Use `ConnectableDevice::getCapability(Class<T> controllerClass)` method instead

VolumeControl getVolumeControl () Accessor for highest priority VolumeControl object This method is deprecated. Use `ConnectableDevice::getCapability(Class<T> controllerClass)` method instead

WebAppLauncher getWebAppLauncher () Accessor for highest priority WebAppLauncher object This method is deprecated. Use `ConnectableDevice::getCapability(Class<T> controllerClass)` method instead

TVControl getTVControl () Accessor for highest priority TVControl object This method is deprecated. Use `ConnectableDevice::getCapability(Class<T> controllerClass)` method instead

ToastControl getToastControl () Accessor for highest priority ToastControl object This method is deprecated. Use `ConnectableDevice::getCapability(Class<T> controllerClass)` method instead

TextInputControl getTextInputControl () Accessor for highest priority TextInputControl object This method is deprecated. Use `ConnectableDevice::getCapability(Class<T> controllerClass)` method instead

MouseControl getMouseControl () Accessor for highest priority MouseControl object This method is deprecated. Use `ConnectableDevice::getCapability(Class<T> controllerClass)` method instead

ExternalInputControl getExternalInputControl () Accessor for highest priority ExternalInputControl object This method is deprecated. Use `ConnectableDevice::getCapability(Class<T> controllerClass)` method instead

PowerControl getPowerControl () Accessor for highest priority PowerLauncher object This method is deprecated. Use `ConnectableDevice::getCapability(Class<T> controllerClass)` method instead

KeyControl getKeyControl () Accessor for highest priority KeyControl object This method is deprecated. Use `ConnectableDevice::getCapability(Class<T> controllerClass)` method instead

void setIpAddress (String *ipAddress*) Sets the IP address of the ConnectableDevice.

Parameters:

- *ipAddress* – IP address of the ConnectableDevice

String getIpAddress () Gets the Current IP address of the ConnectableDevice.

void setFriendlyName (String *friendlyName*) Sets an estimate of the ConnectableDevice's current friendly name.

Parameters:

- *friendlyName* – Friendly name of the device

String getFriendlyName () Gets an estimate of the ConnectableDevice's current friendly name.

void setLastKnownIPAddress (String *lastKnownIPAddress*) Sets the last IP address this ConnectableDevice was discovered at.

Parameters:

- *lastKnownIPAddress* – Last known IP address of the device & it's services

String getLastKnownIPAddress () Gets the last IP address this ConnectableDevice was discovered at.

void setLastSeenOnWifi (String *lastSeenOnWifi*) Sets the name of the last wireless network this ConnectableDevice was discovered on.

Parameters:

- *lastSeenOnWifi* – Last Wi-Fi network this device & it's services were discovered on

String getLastSeenOnWifi () Gets the name of the last wireless network this ConnectableDevice was discovered on.

void setLastConnected (long *lastConnected*) Sets the last time (in milli seconds from 1970) that this ConnectableDevice was connected to.

Parameters:

- *lastConnected* – Last connected time

long getLastConnected () Gets the last time (in milli seconds from 1970) that this ConnectableDevice was connected to.

void setLastDetection (long *lastDetection*) Sets the last time (in milli seconds from 1970) that this ConnectableDevice was detected.

Parameters:

- *lastDetection* – Last detected time

long getLastDetection () Gets the last time (in milli seconds from 1970) that this ConnectableDevice was detected.

void setModelName (String *modelName*) Sets an estimate of the ConnectableDevice's current model name.

Parameters:

- *modelName* – Model name of the ConnectableDevice

String getModelName () Gets an estimate of the ConnectableDevice's current model name.

void setModelNumber (String *modelName*) Sets an estimate of the ConnectableDevice's current model number.

Parameters:

- *modelName* – Model number of the ConnectableDevice

String getModelNumber () Gets an estimate of the ConnectableDevice's current model number.

void setId (String id) Sets the universally unique id of this particular ConnectableDevice object. This is used internally in the SDK and should not be used.

Parameters:

- id – New id for the ConnectableDevice

String getId () Universally unique id of this particular ConnectableDevice object, persists between sessions in ConnectableDeviceStore for connected devices

public<T extends CapabilityMethods> T getCapability (Class<T> controllerClass) Get a capability with the highest priority from a device. If device doesn't have such capability then returns null.

Parameters:

- controllerClass – type of capability

Returns: capability implementation

Inherited Methods

void onConnectionRequired (DeviceService service) If the DeviceService requires an active connection (websocket, pairing, etc) this method will be called.

Parameters:

- service – DeviceService that requires connection

void onConnectionSuccess (DeviceService service) After the connection has been successfully established, and after pairing (if applicable), this method will be called.

Parameters:

- service – DeviceService that was successfully connected

void onCapabilitiesUpdated (DeviceService service, List<String> added, List<String> removed) There are situations in which a DeviceService will update the capabilities it supports and propagate these changes to the DeviceService. Such situations include:

- on discovery, DIALService will reach out to detect if certain apps are installed
- on discovery, certain DeviceServices need to reach out for version & region information

For more information on this particular method, see ConnectableDeviceDelegate's connectableDevice:capabilitiesAdded:removed: method.

Parameters:

- service – DeviceService that has experienced a change in capabilities
- added – List<String> of capabilities that are new to the DeviceService
- removed – List<String> of capabilities that the DeviceService has lost

void onDisconnect (DeviceService service, Error error) This method will be called on any disconnection. If error is nil, then the connection was clean and likely triggered by the responsible DiscoveryProvider or by the user.

Parameters:

- service – DeviceService that disconnected
- error – Error with a description of any errors causing the disconnect. If this value is nil, then the disconnect was clean/expected.

void onConnectionFailure (*DeviceService service*, **Error error**) Will be called if the DeviceService fails to establish a connection.

Parameters:

- service – DeviceService which has failed to connect
- error – Error with a description of the failure

void onPairingRequired (*DeviceService service*, *PairingType pairingType*, *Object pairingData*) If the DeviceService requires pairing, valuable data will be passed to the delegate via this method.

Parameters:

- service – DeviceService that requires pairing
- pairingType – PairingType that the DeviceService requires
- pairingData – Any data that might be required for the pairing process, will usually be nil

void onPairingSuccess (*DeviceService service*)

Parameters:

- service

void onPairingFailed (*DeviceService service*, **Error error**) If there is any error in pairing, this method will be called.

Parameters:

- service – DeviceService that has failed to complete pairing
- error – Error with a description of the failure

ConnectableDeviceListener

`com.connectsdk.device.ConnectableDeviceListener`

ConnectableDeviceListener allows for a class to receive messages about ConnectableDevice connection, disconnect, and update events.

It also serves as a proxy for message handling when connecting and pairing with each of a ConnectableDevice's DeviceServices. Each of the DeviceService proxy methods are optional and would only be useful in a few use cases.

- providing your own UI for the pairing process.
- interacting directly and exclusively with a single type of DeviceService

Methods

void onDeviceReady (*ConnectableDevice device*) A ConnectableDevice sends out a ready message when all of its connectable DeviceServices have been connected and are ready to receive commands.

Parameters:

- device – ConnectableDevice that is ready for commands.

void onDeviceDisconnected (*ConnectableDevice device*) When all of a ConnectableDevice's DeviceServices have become disconnected, the disconnected message is sent.

Parameters:

- device – ConnectableDevice that has been disconnected.

void onPairingRequired (*ConnectableDevice* device, *DeviceService* service, *PairingType* pairingType)
DeviceService listener proxy method.

This method is called when a DeviceService tries to connect and finds out that it requires pairing information from the user.

Parameters:

- device – ConnectableDevice containing the DeviceService
- service – DeviceService that requires pairing
- pairingType – DeviceServicePairingType that the DeviceService requires

void onCapabilityUpdated (*ConnectableDevice* device, *List<String>* added, *List<String>* removed) When a ConnectableDevice finds & loses DeviceServices, that ConnectableDevice will experience a change in its collective capabilities list. When such a change occurs, this message will be sent with arrays of capabilities that were added & removed.

This message will allow you to decide when to stop/start interacting with a ConnectableDevice, based off of its supported capabilities.

Parameters:

- device – ConnectableDevice that has experienced a change in capabilities
- added – List<String> of capabilities that are new to the ConnectableDevice
- removed – List<String> of capabilities that the ConnectableDevice has lost

void onConnectionFailed (*ConnectableDevice* device, *ServiceCommandError* error) This method is called when the connection to the ConnectableDevice has failed.

Parameters:

- device – ConnectableDevice that has failed to connect
- error – ServiceCommandError with a description of the failure

ServiceSubscription

`com.connectsdk.service.command.ServiceSubscription`

Methods

`void unsubscribe ()`

`T addListener (T listener)`

Parameters:

- listener – (optional) T with methods to be called on success or failure

`void removeListener (T listener)`

Parameters:

- listener – (optional) T with methods to be called on success or failure

`List<T> getListeners ()`

5.10.3 Device Services

AirPlayService

`com.connectsdk.service.AirPlayService`

extends DeviceService

AirPlayService provides media playback/control & web app launching (iOS only) capabilities for Apple TV devices.

AirPlay-enabled speakers are not currently supported by Connect SDK.

Properties

`final String X_APPLE_SESSION_ID = "X-Apple-Session-ID"`

`final String ID = "AirPlay"`

Inner Classes

- PlaybackPositionListener

Methods

***CapabilityPriorityLevel* getPriorityLevel (Class<? extends CapabilityMethods > clazz) Parameters:**

- clazz

AirPlayService (ServiceDescription serviceDescription, ServiceConfig serviceConfig) Parameters:

- serviceDescription
- serviceConfig

***MediaControl* getMediaControl ()** Get MediaControl implementation

Returns: MediaControl

***CapabilityPriorityLevel* getMediaControlCapabilityLevel ()** Get a capability priority for current implementation

Returns: CapabilityPriorityLevel

void play (*ResponseListener* <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void pause (*ResponseListener* <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void stop (*ResponseListener* <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void rewind (*ResponseListener* <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void fastForward (*ResponseListener* <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void previous (*ResponseListener* <Object> *listener*) This method is deprecated. Use `PlaylistControl::previous (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void next (*ResponseListener* <Object> *listener*) This method is deprecated. Use `PlaylistControl::next (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void seek (*long position*, *ResponseListener* <Object> *listener*) **Parameters:**

- position – The new position, in milliseconds from the beginning of the stream
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getPosition (*final :doc: PositionListener* <and-positionlistener> *listener*) **Parameters:**

- listener – (optional) `final PositionListener` with methods to be called on success or failure

void getPlayState (*final PlayStateListener* *listener*) AirPlay has the same response for Buffering and Finished states that's why this method always returns Finished state for video which is not ready to play.

Parameters:

- listener – (optional) `final PlayStateListener` with methods to be called on success or failure

void getDuration (*final DurationListener* *listener*) **Parameters:**

- listener – (optional) `final DurationListener` with methods to be called on success or failure

ServiceSubscription <*PlayStateListener*> **subscribePlayState** (*PlayStateListener* *listener*) Subscribe for playback state changes

Parameters:

- listener – receives play state notifications

Returns: `ServiceSubscription<PlayStateListener>`

MediaPlayer **getMediaPlayer** ()

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel** ()

void getMediaInfo (*MediaInfoListener* *listener*) **Parameters:**

- listener – (optional) `MediaInfoListener` with methods to be called on success or failure

ServiceSubscription <*MediaInfoListener*> **subscribeMediaInfo** (*MediaInfoListener* *listener*) **Parameters:**

- listener – (optional) `MediaInfoListener` with methods to be called on success or failure

void displayImage (*final String url*, *String mimeType*, *String title*, *String description*, *String iconSrc*, *final LaunchListener* *listener*)

Parameters:

- url
- mimeType
- title
- description
- iconSrc
- listener – (optional) `final LaunchListener` with methods to be called on success or failure

void displayImage (*MediaInfo* mediaInfo, *LaunchListener* listener) **Parameters:**

- mediaInfo
- listener – (optional) *LaunchListener* with methods to be called on success or failure

void playVideo (final *String* url, *String* mimeType, *String* title, *String* description, *String* iconSrc, boolean shouldLoop, final *LaunchListener* listener) **Parameters:**

- url
- mimeType
- title
- description
- iconSrc
- shouldLoop
- listener – (optional) final *LaunchListener* with methods to be called on success or failure

void playMedia (*String* url, *String* mimeType, *String* title, *String* description, *String* iconSrc, boolean shouldLoop, *LaunchListener* listener) **Parameters:**

This method is deprecated. Use `MediaPlayer::playMedia(MediaInfo mediaInfo, boolean shouldLoop, LaunchListener listener)` instead.

Parameters:

- url
- mimeType
- title
- description
- iconSrc
- shouldLoop
- listener – (optional) *LaunchListener* with methods to be called on success or failure

void playMedia (*MediaInfo* mediaInfo, boolean shouldLoop, *LaunchListener* listener) **Parameters:**

- mediaInfo
- shouldLoop
- listener – (optional) *LaunchListener* with methods to be called on success or failure

void closeMedia (*LaunchSession* launchSession, *ResponseListener* <Object> listener) **Parameters:**

- launchSession
- listener – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void sendCommand (final *ServiceCommand*<?> serviceCommand) **Parameters:**

- serviceCommand

void sendPairingKey (*String* pairingKey) **Parameters:**

- pairingKey

boolean **isConnectable** ()

boolean **isConnected** ()

void **connect** ()

void **disconnect** ()

void **onLoseReachability (DeviceServiceReachability *reachability*)** Parameters:

- reachability

static DiscoveryFilter **discoveryFilter** ()

Inherited Methods

void connect () Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceListener. If the connection attempt reveals that pairing is required, the DeviceServiceListener will also be notified in that event.

void disconnect () Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceListener.

boolean isConnected () Whether the DeviceService is currently connected

boolean **isConnectable** ()

void cancelPairing () Explicitly cancels pairing in services that require pairing. In some services, this will hide a prompt that is displaying on the device.

void sendPairingKey (String *pairingKey*) Will attempt to pair with the DeviceService with the provided pairing-Data. The failure/success will be reported back to the DeviceServiceListener.

Parameters:

- pairingKey – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

List<String> **getCapabilities** ()

boolean hasCapability (String *capability*) Test to see if the capabilities array contains a given capability. See the individual Capability classes for acceptable capability values.

It is possible to append a wildcard search term `.Any` to the end of the search term. This method will return true for capabilities that match the term up to the wildcard.

Example: `Launcher.App.Any`

Parameters:

- capability – Capability to test against

boolean hasAnyCapability (String... *capabilities*) Test to see if the capabilities array contains at least one capability in a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – Set of capabilities to test against

boolean hasCapabilities (List<String> *capabilities*) Test to see if the capabilities array contains a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – List of capabilities to test against

ServiceDescription **getServiceDescription** ()

ServiceConfig **getServiceConfig** ()

JSONObject **toJSONObject** ()

String getServiceName () Name of the DeviceService (webOS, Chromecast, etc)

void closeLaunchSession (*LaunchSession launchSession, ResponseListener <Object> listener*) Closes the session on the first screen device. Depending on the sessionType, the associated service will have different ways of handling the close functionality.

Parameters:

- launchSession – LaunchSession to close
- listener – (optional) listener to be called on success/failure

MediaPlayer **getMediaPlayer** ()

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel** ()

void getMediaInfo (*MediaInfoListener listener*) **Parameters:**

- listener – (optional) MediaInfoListener with methods to be called on success or failure

ServiceSubscription <MediaInfoListener> **subscribeMediaInfo** (*MediaInfoListener listener*) **Parameters:**

- listener – (optional) MediaInfoListener with methods to be called on success or failure

void displayImage (*MediaInfo mediaInfo, LaunchListener listener*) Display an image on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- MediaPlayer.Display.Image
- MediaPlayer.MediaData.Title
- MediaPlayer.MediaData.Description
- MediaPlayer.MediaData.Thumbnail
- MediaPlayer.MediaData.MimeType

Parameters:

- mediaInfo – Object of MediaInfo class which includes all the information about an image to display.
- listener – (optional) LaunchListener with methods to be called on success or failure

void playMedia (*MediaInfo mediaInfo, boolean shouldLoop, LaunchListener listener*) Play an audio or video file on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- MediaPlayer.Play.Video
- MediaPlayer.Play.Audio
- MediaPlayer.MediaData.Title
- MediaPlayer.MediaData.Description
- MediaPlayer.MediaData.Thumbnail
- MediaPlayer.MediaData.MimeType

Parameters:

- `mediaInfo` – Object of `MediaInfo` class which includes all the information about an image to display.
- `shouldLoop` – Whether to automatically loop playback
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void closeMedia (*LaunchSession* `launchSession`, *ResponseListener* `<Object> listener`) Close a running media session. Because media is handled differently on different platforms, it is required to keep track of `LaunchSession` and `MediaControl` objects to control that media session in the future. `LaunchSession` will be required to close the media and `mediaControl` will be required to control the media.

Related capabilities:

- `MediaPlayer.Close`

Parameters:

- `launchSession` – `LaunchSession` object for use in closing media instance
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

MediaControl `getMediaControl ()`

Get `MediaControl` implementation

Returns: `MediaControl`

CapabilityPriorityLevel `getMediaControlCapabilityLevel ()`

Get a capability priority for current implementation

Returns: `CapabilityPriorityLevel`

void play (*ResponseListener* `<Object> listener`) Send play command.

Related capabilities:

- `MediaControl.Play`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void pause (*ResponseListener* `<Object> listener`) Send pause command.

Related capabilities:

- `MediaControl.Pause`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void stop (*ResponseListener* `<Object> listener`) Send play command.

Related capabilities:

- `MediaControl.Stop`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void rewind (*ResponseListener* `<Object> listener`) Send rewind command.

Related capabilities:

- `MediaControl.Rewind`

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void fastForward (*ResponseListener* <Object> listener) Send play command.

Related capabilities:

- `MediaControl.FastForward`

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void previous (*ResponseListener* <Object> listener) This method is deprecated. Use `PlaylistControl::previous(ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void next (*ResponseListener* <Object> listener) This method is deprecated. Use `PlaylistControl::next(ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void seek (long position, *ResponseListener* <Object> listener) Seeks to a new position within the current media item

Related capabilities:

- `MediaControl.Seek`

Parameters:

- position – The new position, in milliseconds from the beginning of the stream
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getDuration (*DurationListener* listener) Get the current media duration in milliseconds

Parameters:

- listener – (optional) `DurationListener` with methods to be called on success or failure

void getPosition (:doc: *PositionListener* <and-positionlistener> listener) Get the current playback position in milliseconds

Parameters:

- listener – (optional) `PositionListener` with methods to be called on success or failure

void getPlayState (*PlayStateListener* listener) Get the current state of playback

Parameters:

- listener – (optional) `PlayStateListener` with methods to be called on success or failure

ServiceSubscription <*PlayStateListener*> **subscribePlayState** (*PlayStateListener* listener) Subscribe for playback state changes

Parameters:

- listener – receives play state notifications

Returns: `ServiceSubscription<PlayStateListener>`

void onLoseReachability (`DeviceServiceReachability` reachability) **Parameters:**

- reachability

void unsubscribe (*URLServiceSubscription*<?> *subscription*) **Parameters:**

- subscription

void sendCommand (*ServiceCommand*<?> *command*) **Parameters:**

- command

CastService

`com.connectsdk.service.CastService`

extends *DeviceService*

CastService provides capabilities for Google Chromecast devices. CastService acts as a layer on top of Google's own Cast SDK, and requires the Cast SDK library to function. CastService provides the following functionality:

- Media playback
- Media control
- Web app launching & two-way communication
- Volume control

Using Connect SDK for discovery/control of Chromecast devices will result in your app complying with the Google Cast SDK [terms of service](#).

To learn more about Cast SDK, visit the [Google Cast SDK Developer site](#).

Inner Classes

- ApplicationConnectionResultCallback
- CastListener
- ConnectionCallbacks
- ConnectionFailedListener
- ConnectionListener
- LaunchWebAppListener

Methods

CastService (*ServiceDescription* *serviceDescription*, *ServiceConfig* *serviceConfig*)

Parameters:

- serviceDescription
- serviceConfig

String **getServiceName** ()

CapabilityPriorityLevel **getPriorityLevel** (*Class*<? extends *CapabilityMethods* > *clazz*) **Parameters:**

- clazz

void **connect** ()

void **disconnect** ()

MediaControl **getMediaControl ()** Get MediaControl implementation

Returns: MediaControl

CapabilityPriorityLevel **getMediaControlCapabilityLevel ()** Get a capability priority for current implementation

Returns: CapabilityPriorityLevel

void play (final *ResponseListener* <Object> listener) Parameters:

- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void pause (final *ResponseListener* <Object> listener) Parameters:

- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void stop (final *ResponseListener* <Object> listener) Parameters:

- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void rewind (*ResponseListener* <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void fastForward (*ResponseListener* <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void previous (*ResponseListener* <Object> listener) This method is deprecated. Use `PlaylistControl::previous (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void next (*ResponseListener* <Object> listener) This method is deprecated. Use `PlaylistControl::next (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void seek (final long position, final *ResponseListener* <Object> listener) Parameters:

- position
- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void getDuration (final *DurationListener* listener) Parameters:

- listener – (optional) final DurationListener with methods to be called on success or failure

void getPosition (final *PositionListener* listener) Parameters:

- listener – (optional) final PositionListener with methods to be called on success or failure

MediaPlayer **getMediaPlayer ()**

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel ()**

void getMediaInfo (*MediaInfoListener* listener) Parameters:

- listener – (optional) MediaInfoListener with methods to be called on success or failure

ServiceSubscription <*MediaInfoListener*> **subscribeMediaInfo (*MediaInfoListener* listener) Parameters:**

- listener – (optional) MediaInfoListener with methods to be called on success or failure

void displayImage (String url, String mimeType, String title, String description, String iconSrc, LaunchListener listener)
 This method is deprecated. Use `MediaPlayer::displayImage(MediaInfo mediaInfo, LaunchListener listener)` instead.

Parameters:

- url
- mimeType
- title
- description
- iconSrc
- listener – (optional) LaunchListener with methods to be called on success or failure

void displayImage (*MediaInfo* mediaInfo, LaunchListener listener) Parameters:

- mediaInfo
- listener – (optional) LaunchListener with methods to be called on success or failure

void playMedia (String url, String mimeType, String title, String description, String iconSrc, boolean shouldLoop, LaunchListener listener)

This method is deprecated. Use `MediaPlayer::playMedia(MediaInfo mediaInfo, boolean shouldLoop, LaunchListener listener)` instead.

Parameters:

- url
- mimeType
- title
- description
- iconSrc
- shouldLoop
- listener – (optional) LaunchListener with methods to be called on success or failure

void playMedia (*MediaInfo* mediaInfo, boolean shouldLoop, LaunchListener listener) Parameters:

- mediaInfo
- shouldLoop
- listener – (optional) LaunchListener with methods to be called on success or failure

void closeMedia (final *LaunchSession* launchSession, final *ResponseListener* <Object> listener) Parameters:

- launchSession
- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

WebAppLauncher **getWebAppLauncher ()**

CapabilityPriorityLevel **getWebAppLauncherCapabilityLevel ()**

void launchWebApp (String webAppId, *WebAppSession.LaunchListener* listener) Parameters:

- webAppId
- listener – (optional) WebAppSession.LaunchListener with methods to be called on success or failure

void launchWebApp (final String webAppId, final boolean relaunchIfRunning, final *WebAppSession.LaunchListener* listener)
Parameters:

- webAppId
- relaunchIfRunning
- listener – (optional) final WebAppSession.LaunchListener with methods to be called on success or failure

void launchWebApp (String webAppId, JSONObject params, WebAppSession.LaunchListener listener)

Parameters:

- webAppId
- params
- listener – (optional) WebAppSession.LaunchListener with methods to be called on success or failure

void launchWebApp (String webAppId, JSONObject params, boolean relaunchIfRunning, WebAppSession.LaunchListener listener)

Parameters:

- webAppId
- params
- relaunchIfRunning
- listener – (optional) WebAppSession.LaunchListener with methods to be called on success or failure

void requestStatus (final ResponseListener <Object> listener) Parameters:

- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void joinApplication (final ResponseListener <Object> listener) Parameters:

- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void joinWebApp (final LaunchSession webAppLaunchSession, final WebAppSession.LaunchListener listener)

Parameters:

- webAppLaunchSession
- listener – (optional) final WebAppSession.LaunchListener with methods to be called on success or failure

void joinWebApp (String webAppId, WebAppSession.LaunchListener listener) Parameters:

- webAppId
- listener – (optional) WebAppSession.LaunchListener with methods to be called on success or failure

void closeWebApp (LaunchSession launchSession, final ResponseListener <Object> listener) Parameters:

- launchSession
- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void pinWebApp (String webAppId, ResponseListener <Object> listener) Parameters:

- webAppId
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void unPinWebApp (String webAppId, ResponseListener <Object> listener) Parameters:

- webAppId
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void isWebAppPinned (String webAppId, WebAppPinStatusListener listener) Parameters:

- webAppId
- listener – (optional) WebAppPinStatusListener with methods to be called on success or failure

ServiceSubscription <WebAppPinStatusListener> **subscribeIsWebAppPinned** (String *webAppId*, *WebAppPinStatusListener listener*)

Parameters:

- webAppId
- listener – (optional) WebAppPinStatusListener with methods to be called on success or failure

VolumeControl **getVolumeControl** ()

CapabilityPriorityLevel **getVolumeControlCapabilityLevel** ()

void volumeUp (final *ResponseListener* <Object> *listener*) **Parameters:**

- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void volumeDown (final *ResponseListener* <Object> *listener*) **Parameters:**

- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void setVolume (final float *volume*, final *ResponseListener* <Object> *listener*) **Parameters:**

- volume
- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void getVolume (*VolumeListener listener*) **Parameters:**

- listener – (optional) VolumeListener with methods to be called on success or failure

void setMute (final boolean *isMute*, final *ResponseListener* <Object> *listener*) **Parameters:**

- isMute
- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void getMute (final *MuteListener listener*) **Parameters:**

- listener – (optional) final MuteListener with methods to be called on success or failure

ServiceSubscription <VolumeListener> **subscribeVolume** (*VolumeListener listener*) **Parameters:**

- listener – (optional) VolumeListener with methods to be called on success or failure

ServiceSubscription <MuteListener> **subscribeMute** (*MuteListener listener*) **Parameters:**

- listener – (optional) MuteListener with methods to be called on success or failure

void getPlayState (*PlayStateListener listener*) Get the current state of playback

Parameters:

- listener – (optional) PlayStateListener with methods to be called on success or failure

ApiClient **getApiClient** ()

boolean **isConnectable** ()

boolean **isConnected** ()

ServiceSubscription <PlayStateListener> **subscribePlayState** (*PlayStateListener listener*) Subscribe for playback state changes

Parameters:

- listener – receives play state notifications

Returns: ServiceSubscription<PlayStateListener>

void unsubscribe (*URLServiceSubscription*<?> *subscription*) **Parameters:**

- subscription

List<URLServiceSubscription<?>> **getSubscriptions** ()

void setSubscriptions (List< URLServiceSubscription<?>> *subscriptions*) **Parameters:**

- subscriptions

static DiscoveryFilter **discoveryFilter** ()

static void setApplicationID (String *id*) **Parameters:**

- id

static String **getApplicationID** ()

Inherited Methods

void connect () Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceListener. If the connection attempt reveals that pairing is required, the DeviceServiceListener will also be notified in that event.

void disconnect () Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceListener.

boolean isConnected () Whether the DeviceService is currently connected

boolean **isConnectable** ()

void cancelPairing () Explicitly cancels pairing in services that require pairing. In some services, this will hide a prompt that is displaying on the device.

void sendPairingKey (String *pairingKey*) Will attempt to pair with the DeviceService with the provided pairing-Data. The failure/success will be reported back to the DeviceServiceListener.

Parameters:

- pairingKey – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

List<String> **getCapabilities** ()

boolean hasCapability (String *capability*) Test to see if the capabilities array contains a given capability. See the individual Capability classes for acceptable capability values.

It is possible to append a wildcard search term `.Any` to the end of the search term. This method will return true for capabilities that match the term up to the wildcard.

Example: `Launcher.App.Any`

Parameters:

- capability – Capability to test against

boolean hasAnyCapability (String... *capabilities*) Test to see if the capabilities array contains at least one capability in a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – Set of capabilities to test against

boolean hasCapabilities (List<String> *capabilities*) Test to see if the capabilities array contains a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – List of capabilities to test against

ServiceDescription **getServiceDescription** ()

ServiceConfig **getServiceConfig** ()

JSONObject **toJSONObject** ()

String getServiceName () Name of the DeviceService (webOS, Chromecast, etc)

void closeLaunchSession (*LaunchSession launchSession*, *ResponseListener <Object> listener*) Closes the session on the first screen device. Depending on the sessionType, the associated service will have different ways of handling the close functionality.

Parameters:

- launchSession – LaunchSession to close
- listener – (optional) listener to be called on success/failure

MediaPlayer **getMediaPlayer** ()

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel** ()

void getMediaInfo (*MediaInfoListener listener*) **Parameters:**

- listener – (optional) MediaInfoListener with methods to be called on success or failure

ServiceSubscription <MediaInfoListener> **subscribeMediaInfo** (*MediaInfoListener listener*) **Parameters:**

- listener – (optional) MediaInfoListener with methods to be called on success or failure

void displayImage (*MediaInfo mediaInfo*, *LaunchListener listener*) Display an image on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- MediaPlayer.Display.Image
- MediaPlayer.MediaData.Title
- MediaPlayer.MediaData.Description
- MediaPlayer.MediaData.Thumbnail
- MediaPlayer.MediaData.MimeType

Parameters:

- mediaInfo – Object of MediaInfo class which includes all the information about an image to display.
- listener – (optional) LaunchListener with methods to be called on success or failure

void playMedia (*MediaInfo mediaInfo*, *boolean shouldLoop*, *LaunchListener listener*) Play an audio or video file on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- MediaPlayer.Play.Video
- MediaPlayer.Play.Audio
- MediaPlayer.MediaData.Title
- MediaPlayer.MediaData.Description
- MediaPlayer.MediaData.Thumbnail

- `MediaPlayer.MediaData.MimeType`

Parameters:

- `mediaInfo` – Object of `MediaInfo` class which includes all the information about an image to display.
- `shouldLoop` – Whether to automatically loop playback
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void closeMedia (*LaunchSession launchSession, ResponseListener <Object> listener*) Close a running media session. Because media is handled differently on different platforms, it is required to keep track of `LaunchSession` and `MediaControl` objects to control that media session in the future. `LaunchSession` will be required to close the media and `mediaControl` will be required to control the media.

Related capabilities:

- `MediaPlayer.Close`

Parameters:

- `launchSession` – `LaunchSession` object for use in closing media instance
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

MediaControl **getMediaControl** () Get `MediaControl` implementation

Returns: `MediaControl`

CapabilityPriorityLevel **getMediaControlCapabilityLevel** () Get a capability priority for current implementation

Returns: `CapabilityPriorityLevel`

void play (*ResponseListener <Object> listener*) Send play command.

Related capabilities:

- `MediaControl.Play`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void pause (*ResponseListener <Object> listener*) Send pause command.

Related capabilities:

- `MediaControl.Pause`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void stop (*ResponseListener <Object> listener*) Send play command.

Related capabilities:

- `MediaControl.Stop`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void rewind (*ResponseListener <Object> listener*) Send rewind command.

Related capabilities:

- `MediaControl.Rewind`

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void fastForward (*ResponseListener* <Object> listener) Send play command.

Related capabilities:

- `MediaControl.FastForward`

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void previous (*ResponseListener* <Object> listener) This method is deprecated. Use `PlaylistControl::previous (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void next (*ResponseListener* <Object> listener) This method is deprecated. Use `PlaylistControl::next (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void seek (long position, *ResponseListener* <Object> listener) Seeks to a new position within the current media item

Related capabilities:

- `MediaControl.Seek`

Parameters:

- position – The new position, in milliseconds from the beginning of the stream
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getDuration (*DurationListener* listener) Get the current media duration in milliseconds

Parameters:

- listener – (optional) `DurationListener` with methods to be called on success or failure

void getPosition (*PositionListener* listener) Get the current playback position in milliseconds

Parameters:

- listener – (optional) `PositionListener` with methods to be called on success or failure

void getPlayState (*PlayStateListener* listener) Get the current state of playback

Parameters:

- listener – (optional) `PlayStateListener` with methods to be called on success or failure

ServiceSubscription <*PlayStateListener*> **subscribePlayState** (*PlayStateListener* listener) Subscribe for playback state changes

Parameters:

- listener – receives play state notifications

Returns: `ServiceSubscription<PlayStateListener>`

VolumeControl **getVolumeControl** ()

CapabilityPriorityLevel **getVolumeControlCapabilityLevel** ()

void volumeUp (*ResponseListener* <Object> *listener*) Sends the volume up command to the device.

Related capabilities:

- `VolumeControl.UpDown`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void volumeDown (*ResponseListener* <Object> *listener*) Sends the volume down command to the device.

Related capabilities:

- `VolumeControl.UpDown`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void setVolume (`float volume`, *ResponseListener* <Object> *listener*) Set the volume of the device.

Related capabilities:

- `VolumeControl.Set`

Parameters:

- *volume* – Volume as a float between 0.0 and 1.0
- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getVolume (*VolumeListener* *listener*) Get the current volume of the device.

Related capabilities:

- `VolumeControl.Get`

Parameters:

- *listener* – (optional) `VolumeListener` with methods to be called on success or failure

void setMute (`boolean isMute`, *ResponseListener* <Object> *listener*) Set the current volume.

Related capabilities:

- `VolumeControl.Mute.Set`

Parameters:

- *isMute*
- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getMute (*MuteListener* *listener*) Get the current mute state.

Related capabilities:

- `VolumeControl.Mute.Get`

Parameters:

- *listener* – (optional) `MuteListener` with methods to be called on success or failure

ServiceSubscription <*VolumeListener*> **subscribeVolume** (*VolumeListener* *listener*) Subscribe to the volume on the TV.

Related capabilities:

- `VolumeControl.Subscribe`

Parameters:

- listener – (optional) VolumeListener with methods to be called on success or failure

ServiceSubscription *<MuteListener>* **subscribeMute** (*MuteListener* listener) Subscribe to the mute state on the TV.

Related capabilities:

- `VolumeControl.Mute.Subscribe`

Parameters:

- listener – (optional) MuteListener with methods to be called on success or failure

WebAppLauncher **getWebAppLauncher** ()

CapabilityPriorityLevel **getWebAppLauncherCapabilityLevel** ()

void launchWebApp (String *webAppId*, *LaunchListener* listener) Launch a web application on the TV.

Related capabilities:

- `WebAppLauncher.Launch`
- `WebAppLauncher.Launch.Params` – if launching with params

Parameters:

- *webAppId* – ID of web app assigned by platform vendor
- listener – (optional) LaunchListener with methods to be called on success or failure

void joinWebApp (*LaunchSession* *webAppLaunchSession*, *LaunchListener* listener) Join an active web app without launching/relaunching. If the app is not running/joinable, the failure block will be called immediately.

Related capabilities:

- `WebAppLauncher.Send`
- `WebAppLauncher.Receive`

Parameters:

- *webAppLaunchSession* – LaunchSession for the web app to be joined
- listener – (optional) LaunchListener with methods to be called on success or failure

void closeWebApp (*LaunchSession* *launchSession*, *ResponseListener* *<Object>* listener) Closes a web app with the provided LaunchSession.

Related capabilities:

- `WebAppLauncher.Close`

Parameters:

- *launchSession* – LaunchSession associated with the web app to be closed
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void pinWebApp (String *webAppId*, *ResponseListener* *<Object>* listener) **Parameters:**

- *webAppId*
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void unPinWebApp (String *webAppId*, *ResponseListener* *<Object>* listener) **Parameters:**

- *webAppId*
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void isWebAppPinned (String *webAppId*, *WebAppPinStatusListener* listener) Parameters:

- webAppId
- listener – (optional) WebAppPinStatusListener with methods to be called on success or failure

ServiceSubscription <*WebAppPinStatusListener*> **subscribeIsWebAppPinned (String *webAppId*, *WebAppPinStatusListener* listener)**

Parameters:

- webAppId
- listener – (optional) WebAppPinStatusListener with methods to be called on success or failure

void onLoseReachability (DeviceServiceReachability *reachability*) Parameters:

- reachability

void unsubscribe (URLServiceSubscription<?> *subscription*) Parameters:

- subscription

void sendCommand (ServiceCommand<?> *command*) Parameters:

- command

DIALService

com.connectsdk.service.DIALService

extends DeviceService

DIALService is a full implementation of the Discover And Launch (DIAL) protocol specification. DIALService is used to launch & close apps on DIAL-enabled devices. It can also be used to probe for an app's existence on a DIAL-enabled device. DIAL commands occur over HTTP.

See the [DIAL protocol specification](#) for more information.

Properties

final String ID = "DIAL"

Methods

static void registerApp (String *appId*) Registers an app ID to be checked upon discovery of this device. If the app is found on the target device, the DIALService will gain the "Launcher." capability, where is the value of the *appId* parameter.

This method must be called before starting DiscoveryManager for the first time.

Parameters:

- *appId* - ID of the app to be checked for

static DiscoveryFilter **discoveryFilter** ()

DIALService (ServiceDescription *serviceDescription*, ServiceConfig *serviceConfig*) Parameters:

- *serviceDescription*
- *serviceConfig*

CapabilityPriorityLevel **getPriorityLevel (Class<? extends CapabilityMethods > *clazz*) Parameters:**

- clazz

void setDescription (ServiceDescription serviceDescription) Parameters:

- serviceDescription

Launcher **getLauncher ()**

CapabilityPriorityLevel **getLauncherCapabilityLevel ()**

void launchApp (String appId, AppLaunchListener listener) Parameters:

- appId
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void launchAppWithInfo (AppInfo appInfo, AppLaunchListener listener) Parameters:

- appInfo
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void launchAppWithInfo (final AppInfo appInfo, Object params, final AppLaunchListener listener) Parameters:

- appInfo
- params
- listener – (optional) final AppLaunchListener with methods to be called on success or failure

void launchBrowser (String url, AppLaunchListener listener) Parameters:

- url
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void closeApp (final LaunchSession launchSession, final ResponseListener <Object> listener) Parameters:

- launchSession
- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void launchYouTube (String contentId, AppLaunchListener listener) Parameters:

- contentId
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void launchYouTube (String contentId, float startTime, AppLaunchListener listener) Parameters:

- contentId
- startTime
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void launchHulu (String contentId, AppLaunchListener listener) Parameters:

- contentId
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void launchNetflix (final String contentId, AppLaunchListener listener) Parameters: - contentId - listener – (optional) AppLaunchListener with methods to be called on success or failure

void launchAppStore (String appId, AppLaunchListener listener) Parameters:

- appId
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void getAppList (*AppListListener listener*) **Parameters:**

- listener – (optional) AppListListener with methods to be called on success or failure

void getRunningApp (*AppInfoListener listener*) **Parameters:**

- listener – (optional) AppInfoListener with methods to be called on success or failure

ServiceSubscription <*AppInfoListener*> **subscribeRunningApp** (*AppInfoListener listener*) **Parameters:**

- listener – (optional) AppInfoListener with methods to be called on success or failure

void getAppState (*LaunchSession launchSession*, *AppStateListener listener*) **Parameters:**

- launchSession
- listener – (optional) AppStateListener with methods to be called on success or failure

ServiceSubscription <*AppStateListener*> **subscribeAppState** (*LaunchSession launchSession*, *com.connectsdk.service.capability.Launcher.AppStateListener listener*) **Parameters:**

- launchSession
- listener – (optional) com.connectsdk.service.capability.Launcher.AppStateListener with methods to be called on success or failure

void closeLaunchSession (*LaunchSession launchSession*, *ResponseListener* <*Object*> *listener*) **Parameters:**

- launchSession
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

boolean **isConnectable** ()

boolean **isConnected** ()

void **connect** ()

void **disconnect** ()

void onLoseReachability (*DeviceServiceReachability reachability*) **Parameters:**

- reachability

void sendCommand (*final ServiceCommand*<?> *mCommand*) **Parameters:**

- mCommand

Inherited Methods

void connect () Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceListener. If the connection attempt reveals that pairing is required, the DeviceServiceListener will also be notified in that event.

void disconnect () Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceListener.

boolean isConnected () Whether the DeviceService is currently connected

boolean **isConnectable** ()

void cancelPairing () Explicitly cancels pairing in services that require pairing. In some services, this will hide a prompt that is displaying on the device.

void sendPairingKey (String *pairingKey*) Will attempt to pair with the DeviceService with the provided pairing-Data. The failure/success will be reported back to the DeviceServiceListener.

Parameters:

- *pairingKey* – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

List<String> **getCapabilities ()**

boolean hasCapability (String *capability*) Test to see if the capabilities array contains a given capability. See the individual Capability classes for acceptable capability values.

It is possible to append a wildcard search term `.Any` to the end of the search term. This method will return true for capabilities that match the term up to the wildcard.

Example: `Launcher.App.Any`

Parameters:

- *capability* – Capability to test against

boolean hasAnyCapability (String... *capabilities*) Test to see if the capabilities array contains at least one capability in a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- *capabilities* – Set of capabilities to test against

boolean hasCapabilities (List<String> *capabilities*) Test to see if the capabilities array contains a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- *capabilities* – List of capabilities to test against

ServiceDescription **getServiceDescription ()**

ServiceConfig **getServiceConfig ()**

JSONObject **toJSONObject ()**

String **getServiceName ()** Name of the DeviceService (webOS, Chromecast, etc)

void closeLaunchSession (LaunchSession *launchSession*, ResponseListener <Object> *listener*) Closes the session on the first screen device. Depending on the sessionType, the associated service will have different ways of handling the close functionality.

Parameters:

- *launchSession* – LaunchSession to close
- *listener* – (optional) listener to be called on success/failure

Launcher **getLauncher ()**

CapabilityPriorityLevel **getLauncherCapabilityLevel ()**

void launchAppWithInfo (AppInfo *appInfo*, AppLaunchListener *listener*) Launch an application on the device.

Related capabilities:

- `Launcher.App`
- `Launcher.App.Params` – if launching with params

Parameters:

- appInfo – AppInfo object for the application
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void launchApp (String appId, AppLaunchListener listener) Launch an application on the device.

Related capabilities:

- Launcher.App

Parameters:

- appId – ID of the application
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void closeApp (LaunchSession launchSession, ResponseListener <Object> listener) Close an application on the device.

Related capabilities:

- Launcher.App.Close

Parameters:

- launchSession – LaunchSession of the target app
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void getAppList (AppListListener listener) Gets a list of all apps installed on the device.

Related capabilities:

- Launcher.App.List

Parameters:

- listener – (optional) AppListListener with methods to be called on success or failure

void getRunningApp (AppInfoListener listener) Gets an AppInfo object for the current running app on the device.

Related capabilities:

- Launcher.RunningApp

Parameters:

- listener – (optional) AppInfoListener with methods to be called on success or failure

ServiceSubscription <AppInfoListener> subscribeRunningApp (AppInfoListener listener) Subscribes to changes of the current running app. Every time the running app changes, the success block will be called with an AppInfo object for the current running app.

Related capabilities:

- Launcher.RunningApp.Subscribe

Parameters:

- listener – (optional) AppInfoListener with methods to be called on success or failure

void getAppState (LaunchSession launchSession, AppStateListener listener) Gets the target app's running status and on-screen visibility.

Related capabilities:

- Launcher.AppState

Parameters:

- `launchSession` – `LaunchSession` of the target app
- `listener` – (optional) `AppStateListener` with methods to be called on success or failure

ServiceSubscription `<AppStateListener> subscribeAppState (LaunchSession launchSession, AppStateListener listener)`

Subscribes to changes of the state of the target app. Every time the app's state changes, the success block will be called with info on the app's running status and on-screen visibility.

Related capabilities:

- `Launcher.AppState.Subscribe`

Parameters:

- `launchSession` – `LaunchSession` of the target app
- `listener` – (optional) `AppStateListener` with methods to be called on success or failure

void launchBrowser (String url, AppLaunchListener listener) Launch the web browser. Will launch deep-linked to provided URL, if supported on the target platform.

Related capabilities:

- `Launcher.Browser`
- `Launcher.Browser.Params` – if launching with url

Parameters:

- `url`
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void launchYouTube (String contentId, AppLaunchListener listener) Launch YouTube app. Will launch deep-linked to provided contentId, if supported on the target platform.

Related capabilities:

- `Launcher.YouTube`
- `Launcher.YouTube.Params` – if launching with contentId

Parameters:

- `contentId` – Video id to open
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void launchNetflix (String contentId, AppLaunchListener listener) Launch Netflix app. Will launch deep-linked to provided contentId, if supported on the target platform.

Related capabilities:

- `Launcher.Netflix`
- `Launcher.Netflix.Params` – if launching with contentId

Parameters:

- `contentId` – Video id to open
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void launchHulu (String contentId, AppLaunchListener listener) Launch Hulu app. Will launch deep-linked to provided contentId, if supported on the target platform.

Related capabilities:

- `Launcher.Hulu`
- `Launcher.Hulu.Params` – if launching with `contentId`

Parameters:

- `contentId` – Video id to open
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void launchAppStore (String *appId*, *AppLaunchListener* listener) Launch the device’s app store app, optionally deep-linked to a specific app’s page.

Related capabilities:

- `Launcher.AppStore`
- `Launcher.AppStore.Params`

Parameters:

- `appId` – (optional) ID of the application to show in the app store
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void onLoseReachability (DeviceServiceReachability *reachability*) Parameters:

- `reachability`

void unsubscribe (URLServiceSubscription<?> *subscription*) Parameters:

- `subscription`

void sendCommand (ServiceCommand<?> *command*) Parameters:

- `command`

DLNASService

`com.connectsdk.service.DLNASService`

extends DeviceService

DLNASService is a rough control implementation for the UPnP AVTransport, MediaRenderer, and RenderingControl services. DLNA commands & events occur over HTTP.

This service currently exists for the sole purpose of providing media control/playback functionality for the Net-castTVService. `DiscoveryManager` is currently set up to ignore any DLNA devices that are not manufactured by LG. It is not recommended to remove this restriction, as the DLNASService implementation is not complete.

To learn more about the protocols in use by DLNASService, check out the following documents.

- [UPnP](#)
- [AVTransport Service](#)
- [MediaRenderer Device](#)
- [RenderingControl Service](#)

Properties

```
final String ID = "DLNA" final String AV_TRANSPORT_URN = "urn:schemas-upnp-org:service:AVTransport:1" final String CONNECTION_MANAGER_URN = "urn:schemas-upnp-org:service:ConnectionManager:1" final String RENDERING_CONTROL_URN = "urn:schemas-upnp-org:service:RenderingControl:1" final String PLAY_STATE = "playState" final String DEFAULT_SUBTITLE_MIMETYPE = "text/srt" final String DEFAULT_SUBTITLE_TYPE = "srt"
```

Inner Classes

- PositionInfoListener

Methods

DLNASService (ServiceDescription *serviceDescription*, ServiceConfig *serviceConfig*) Parameters:

- serviceDescription
- serviceConfig

DLNASService (ServiceDescription *serviceDescription*, ServiceConfig *serviceConfig*, Context *context*, DLNAHttpServer *dlnaServer*) Parameters:

- serviceDescription
- serviceConfig
- context
- dlnaServer

***CapabilityPriorityLevel* getPriorityLevel (Class<? extends *CapabilityMethods* > *clazz*) Parameters:**

- *clazz*

void setServiceDescription (ServiceDescription *serviceDescription*) Parameters:

- serviceDescription

***MediaPlayer* getMediaPlayer ()**

***CapabilityPriorityLevel* getMediaPlayerCapabilityLevel ()**

void getMediaInfo (final *MediaInfoListener* *listener*) Parameters:

- listener – (optional) final *MediaInfoListener* with methods to be called on success or failure

***ServiceSubscription* <*MediaInfoListener*> subscribeMediaInfo (*MediaInfoListener* *listener*) Parameters:**

- listener – (optional) *MediaInfoListener* with methods to be called on success or failure

void displayMedia (String *url*, String *mimeType*, String *title*, String *description*, String *iconSrc*, final *LaunchListener* *listener*) Parameters:

- url
- mimeType
- title
- description
- iconSrc

- listener – (optional) final `LaunchListener` with methods to be called on success or failure

void displayImage (String url, String mimeType, String title, String description, String iconSrc, LaunchListener listener)

This method is deprecated. Use `MediaPlayer::displayImage(MediaInfo mediaInfo, LaunchListener listener)` instead.

Parameters:

- url
- mimeType
- title
- description
- iconSrc
- listener – (optional) `LaunchListener` with methods to be called on success or failure

void displayImage (MediaInfo mediaInfo, LaunchListener listener) Parameters:

- mediaInfo
- listener – (optional) `LaunchListener` with methods to be called on success or failure

void playMedia (String url, String mimeType, String title, String description, String iconSrc, boolean shouldLoop, LaunchListener listener)

This method is deprecated. Use `MediaPlayer::playMedia(MediaInfo mediaInfo, boolean shouldLoop, LaunchListener listener)` instead.

Parameters:

- url
- mimeType
- title
- description
- iconSrc
- shouldLoop
- listener – (optional) `LaunchListener` with methods to be called on success or failure

void playMedia (MediaInfo mediaInfo, boolean shouldLoop, LaunchListener listener) Parameters:

- mediaInfo
- shouldLoop
- listener – (optional) `LaunchListener` with methods to be called on success or failure

void closeMedia (LaunchSession launchSession, ResponseListener <Object> listener) Parameters:

- launchSession
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

MediaControl getMediaControl () Get `MediaControl` implementation

Returns: `MediaControl`

CapabilityPriorityLevel getMediaControlCapabilityLevel () Get a capability priority for current implementation

Returns: `CapabilityPriorityLevel`

void play (ResponseListener <Object> listener) Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void pause (*ResponseListener* <Object> listener) **Parameters:**

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void stop (*ResponseListener* <Object> listener) **Parameters:**

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void rewind (*ResponseListener* <Object> listener) **Parameters:**

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void fastForward (*ResponseListener* <Object> listener) **Parameters:**

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

PlaylistControl **getPlaylistControl** ()

CapabilityPriorityLevel **getPlaylistControlCapabilityLevel** ()

void previous (*ResponseListener* <Object> listener) This method is deprecated. Use `PlaylistControl::previous (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void next (*ResponseListener* <Object> listener) This method is deprecated. Use `PlaylistControl::next (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void jumpToTrack (long index, *ResponseListener* <Object> listener) Play a track specified by index in the playlist

Parameters:

- index – index in the playlist, it starts from zero like index of array
- listener – optional response listener

void setPlayMode (*PlayMode* playMode, *ResponseListener* <Object> listener) Set order of playing tracks

Parameters:

- playMode
- listener – optional response listener

void seek (long position, *ResponseListener* <Object> listener) **Parameters:**

- position – The new position, in milliseconds from the beginning of the stream
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getDuration (final *DurationListener* listener) **Parameters:**

- listener – (optional) final `DurationListener` with methods to be called on success or failure

void getPosition (final *PositionListener* listener) **Parameters:**

- listener – (optional) final `PositionListener` with methods to be called on success or failure

void sendCommand (final `ServiceCommand<?>` mCommand) **Parameters:**

- mCommand

LaunchSession **decodeLaunchSession** (String type, `JSONObject` sessionObj) **Parameters:**

- type
- sessionObj

void getPlayState (final *PlayStateListener* listener) Parameters:

- listener – (optional) final *PlayStateListener* with methods to be called on success or failure

ServiceSubscription <*PlayStateListener*> **subscribePlayState (*PlayStateListener* listener)** Subscribe for playback state changes

Parameters:

- listener – receives play state notifications

Returns: *ServiceSubscription*<*PlayStateListener*>

void unsubscribe (URLServiceSubscription<?> subscription) Parameters:

- subscription

boolean **isConnectable** ()

boolean **isConnected** ()

void **connect** ()

void **disconnect** ()

void onLoseReachability (DeviceServiceReachability reachability) Parameters:

- reachability

void **subscribeServices** ()

void **resubscribeServices** ()

void **unsubscribeServices** ()

VolumeControl **getVolumeControl** ()

CapabilityPriorityLevel **getVolumeControlCapabilityLevel** ()

void volumeUp (final *ResponseListener* <Object> listener) Parameters:

- listener – (optional) final *ResponseListener*< Object > with methods to be called on success or failure

void volumeDown (final *ResponseListener* <Object> listener) Parameters:

- listener – (optional) final *ResponseListener*< Object > with methods to be called on success or failure

void setVolume (float volume, *ResponseListener* <Object> listener) Parameters:

- volume
- listener – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void getVolume (final *VolumeListener* listener) Parameters:

- listener – (optional) final *VolumeListener* with methods to be called on success or failure

void setMute (boolean isMute, *ResponseListener* <Object> listener) Parameters:

- isMute
- listener – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void getMute (final *MuteListener* listener) Parameters:

- listener – (optional) final *MuteListener* with methods to be called on success or failure

ServiceSubscription <*VolumeListener*> **subscribeVolume** (*VolumeListener listener*) **Parameters:**

- listener – (optional) VolumeListener with methods to be called on success or failure

ServiceSubscription <*MuteListener*> **subscribeMute** (*MuteListener listener*) **Parameters:**

- listener – (optional) MuteListener with methods to be called on success or failure

static DiscoveryFilter **discoveryFilter** ()

Inherited Methods

void connect () Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceListener. If the connection attempt reveals that pairing is required, the DeviceServiceListener will also be notified in that event.

void disconnect () Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceListener.

boolean isConnected () Whether the DeviceService is currently connected

boolean **isConnectable** ()

void cancelPairing () Explicitly cancels pairing in services that require pairing. In some services, this will hide a prompt that is displaying on the device.

void sendPairingKey (**String pairingKey**) Will attempt to pair with the DeviceService with the provided pairing-Data. The failure/success will be reported back to the DeviceServiceListener.

Parameters:

- pairingKey – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

List<String> **getCapabilities** ()

boolean hasCapability (**String capability**) Test to see if the capabilities array contains a given capability. See the individual Capability classes for acceptable capability values.

It is possible to append a wildcard search term `.Any` to the end of the search term. This method will return true for capabilities that match the term up to the wildcard.

Example: `Launcher.App.Any`

Parameters:

- capability – Capability to test against

boolean hasAnyCapability (**String... capabilities**) Test to see if the capabilities array contains at least one capability in a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – Set of capabilities to test against

boolean hasCapabilities (**List<String> capabilities**) Test to see if the capabilities array contains a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – List of capabilities to test against

ServiceDescription **getServiceDescription** ()

ServiceConfig **getServiceConfig** ()

JSONObject **toJSONObject** ()

String **getServiceName** () Name of the DeviceService (webOS, Chromecast, etc)

void closeLaunchSession (*LaunchSession* **launchSession**, *ResponseListener* **<Object> listener**) Closes the session on the first screen device. Depending on the sessionType, the associated service will have different ways of handling the close functionality.

Parameters:

- launchSession – LaunchSession to close
- listener – (optional) listener to be called on success/failure

PlaylistControl **getPlaylistControl** ()

CapabilityPriorityLevel **getPlaylistControlCapabilityLevel** ()

void previous (*ResponseListener* **<Object> listener**) Jump playlist to the previous track.

Play previous track in the playlist

Related capabilities:

- `PlaylistControl.Previous`

Parameters:

- listener – optional response listener

void next (*ResponseListener* **<Object> listener**) Jump playlist to the next track.

Play next track in the playlist

Related capabilities:

- `PlaylistControl.Next`

Parameters:

- listener – optional response listener

void jumpToTrack (**long index**, *ResponseListener* **<Object> listener**) Jump the playlist to the designated track.

Play a track specified by index in the playlist

Related capabilities:

- `PlaylistControl.JumpToTrack`

Parameters:

- index – index in the playlist, it starts from zero like index of array
- listener – optional response listener

void setPlayMode (*PlayMode* **playMode**, *ResponseListener* **<Object> listener**) Set order of playing tracks

Parameters:

- playMode
- listener – optional response listener

MediaControl **getMediaControl** () Get MediaControl implementation

Returns: MediaControl

CapabilityPriorityLevel getMediaControlCapabilityLevel () Get a capability priority for current implementation

Returns: CapabilityPriorityLevel

void play (*ResponseListener* <Object> *listener*) Send play command.

Related capabilities:

- `MediaControl.Play`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void pause (*ResponseListener* <Object> *listener*) Send pause command.

Related capabilities:

- `MediaControl.Pause`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void stop (*ResponseListener* <Object> *listener*) Send play command.

Related capabilities:

- `MediaControl.Stop`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void rewind (*ResponseListener* <Object> *listener*) Send rewind command.

Related capabilities:

- `MediaControl.Rewind`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void fastForward (*ResponseListener* <Object> *listener*) Send play command.

Related capabilities:

- `MediaControl.FastForward`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void seek (long *position*, *ResponseListener* <Object> *listener*) Seeks to a new position within the current media item

Related capabilities:

- `MediaControl.Seek`

Parameters:

- *position* – The new position, in milliseconds from the beginning of the stream
- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getDuration (*DurationListener* *listener*) Get the current media duration in milliseconds

Parameters:

- listener – (optional) DurationListener with methods to be called on success or failure

void getPosition (*PositionListener listener*) Get the current playback position in milliseconds

Parameters:

- listener – (optional) PositionListener with methods to be called on success or failure

void getPlayState (*PlayStateListener listener*) Get the current state of playback

Parameters:

- listener – (optional) PlayStateListener with methods to be called on success or failure

ServiceSubscription <*PlayStateListener*> **subscribePlayState** (*PlayStateListener listener*) Subscribe for playback state changes

Parameters:

- listener – receives play state notifications

Returns: ServiceSubscription<PlayStateListener>

MediaPlayer **getMediaPlayer** ()

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel** ()

void getMediaInfo (*MediaInfoListener listener*) **Parameters:**

- listener – (optional) MediaInfoListener with methods to be called on success or failure

ServiceSubscription <*MediaInfoListener*> **subscribeMediaInfo** (*MediaInfoListener listener*) **Parameters:**

- listener – (optional) MediaInfoListener with methods to be called on success or failure

void displayImage (*MediaInfo mediaInfo*, *LaunchListener listener*) Display an image on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- MediaPlayer.Display.Image
- MediaPlayer.MediaData.Title
- MediaPlayer.MediaData.Description
- MediaPlayer.MediaData.Thumbnail
- MediaPlayer.MediaData.MimeType

Parameters:

- mediaInfo – Object of MediaInfo class which includes all the information about an image to display.
- listener – (optional) LaunchListener with methods to be called on success or failure

void playMedia (*MediaInfo mediaInfo*, *boolean shouldLoop*, *LaunchListener listener*) Play an audio or video file on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- MediaPlayer.Play.Video
- MediaPlayer.Play.Audio
- MediaPlayer.MediaData.Title
- MediaPlayer.MediaData.Description
- MediaPlayer.MediaData.Thumbnail

- `MediaPlayer.MediaData.MimeType`

Parameters:

- `mediaInfo` – Object of `MediaInfo` class which includes all the information about an image to display.
- `shouldLoop` – Whether to automatically loop playback
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void closeMedia (*LaunchSession launchSession, ResponseListener <Object> listener*) Close a running media session. Because media is handled differently on different platforms, it is required to keep track of `LaunchSession` and `MediaControl` objects to control that media session in the future. `LaunchSession` will be required to close the media and `mediaControl` will be required to control the media.

Related capabilities:

- `MediaPlayer.Close`

Parameters:

- `launchSession` – `LaunchSession` object for use in closing media instance
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

VolumeControl **getVolumeControl** ()

CapabilityPriorityLevel **getVolumeControlCapabilityLevel** ()

void volumeUp (*ResponseListener <Object> listener*) Sends the volume up command to the device.

Related capabilities:

- `VolumeControl.UpDown`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void volumeDown (*ResponseListener <Object> listener*) Sends the volume down command to the device.

Related capabilities:

- `VolumeControl.UpDown`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void setVolume (*float volume, ResponseListener <Object> listener*) Set the volume of the device.

Related capabilities:

- `VolumeControl.Set`

Parameters:

- `volume` – Volume as a float between 0.0 and 1.0
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getVolume (*VolumeListener listener*) Get the current volume of the device.

Related capabilities:

- `VolumeControl.Get`

Parameters:

- `listener` – (optional) `VolumeListener` with methods to be called on success or failure

void setMute (boolean isMute, *ResponseListener* <Object> listener) Set the current volume.

Related capabilities:

- `VolumeControl.Mute.Set`

Parameters:

- isMute
- listener – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void getMute (*MuteListener* listener) Get the current mute state.

Related capabilities:

- `VolumeControl.Mute.Get`

Parameters:

- listener – (optional) *MuteListener* with methods to be called on success or failure

***ServiceSubscription* <*VolumeListener*> subscribeVolume (*VolumeListener* listener)** Subscribe to the volume on the TV.

Related capabilities:

- `VolumeControl.Subscribe`

Parameters:

- listener – (optional) *VolumeListener* with methods to be called on success or failure

***ServiceSubscription* <*MuteListener*> subscribeMute (*MuteListener* listener)** Subscribe to the mute state on the TV.

Related capabilities:

- `VolumeControl.Mute.Subscribe`

Parameters:

- listener – (optional) *MuteListener* with methods to be called on success or failure

void onLoseReachability (*DeviceServiceReachability* reachability) Parameters:

- reachability

void unsubscribe (*URLServiceSubscription*<?> subscription) Parameters:

- subscription

void sendCommand (*ServiceCommand*<?> command) Parameters:

- command

DeviceService

`com.connectsdk.service.DeviceService`

Overview

From a high-level perspective, *DeviceService* completely abstracts the functionality of a particular service/protocol (webOS TV, Netcast TV, Chromecast, Roku, DIAL, etc).

In Depth

DeviceService is an abstract class that is meant to be extended. You shouldn't ever use DeviceService directly, unless extending it to provide support for an additional service/protocol.

Immediately after discovery of a DeviceService, DiscoveryManager will set the DeviceService's Listener to the ConnectableDevice that owns the DeviceService. You should not change the Listener unless you intend to manage the lifecycle of that service. The DeviceService will proxy all of its Listener method calls through the ConnectableDevice's ConnectableDeviceListener.

Connection & Pairing

Your ConnectableDevice object will let you know if you need to connect or pair to any services.

Capabilities

All DeviceService objects have a group of capabilities. These capabilities can be implemented by any object, and that object will be returned when you call the DeviceService's capability methods (launcher, mediaPlayer, volumeControl, etc).

Inner Classes

- *DeviceServiceListener*
- *PairingType*

Methods

void connect () Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceListener. If the connection attempt reveals that pairing is required, the DeviceServiceListener will also be notified in that event.

void disconnect () Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceListener.

boolean isConnected () Whether the DeviceService is currently connected

boolean isConnectable ()

void cancelPairing () Explicitly cancels pairing in services that require pairing. In some services, this will hide a prompt that is displaying on the device.

void sendPairingKey (String *pairingKey*) Will attempt to pair with the DeviceService with the provided pairing-Data. The failure/success will be reported back to the DeviceServiceListener.

Parameters:

- *pairingKey* – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

List<String> **getCapabilities ()**

boolean hasCapability (String *capability*) Test to see if the capabilities array contains a given capability. See the individual Capability classes for acceptable capability values.

It is possible to append a wildcard search term `.Any` to the end of the search term. This method will return true for capabilities that match the term up to the wildcard.

Example: `Launcher.App.Any`

Parameters:

- `capability` – Capability to test against

boolean hasAnyCapability (String... capabilities) Test to see if the capabilities array contains at least one capability in a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- `capabilities` – Set of capabilities to test against

boolean hasCapabilities (List<String> capabilities) Test to see if the capabilities array contains a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- `capabilities` – List of capabilities to test against

boolean hasCapabilities (String... capabilities) Test to see if the capabilities array contains a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- `capabilities` – Set of capabilities to test against

ServiceDescription **getServiceDescription ()**

ServiceConfig **getServiceConfig ()**

JSONObject **toJSONObject ()**

String **getServiceName ()** Name of the DeviceService (webOS, Chromecast, etc)

void closeLaunchSession (LaunchSession launchSession, ResponseListener <Object> listener) Closes the session on the first screen device. Depending on the sessionType, the associated service will have different ways of handling the close functionality.

Parameters:

- `launchSession` – LaunchSession to close
- `listener` – (optional) listener to be called on success/failure

Inherited Methods

void onLoseReachability (DeviceServiceReachability reachability) Parameters:

- `reachability`

void unsubscribe (URLServiceSubscription<?> subscription) Parameters:

- `subscription`

void sendCommand (ServiceCommand<?> command) Parameters:

- `command`

DeviceServiceListener

`com.connectsdk.service.DeviceService.DeviceServiceListener`

Methods

void onConnectionRequired (*DeviceService service*) If the DeviceService requires an active connection (web-socket, pairing, etc) this method will be called.

Parameters:

- service – DeviceService that requires connection

void onConnectionSuccess (*DeviceService service*) After the connection has been successfully established, and after pairing (if applicable), this method will be called.

Parameters:

- service – DeviceService that was successfully connected

void onCapabilitiesUpdated (*DeviceService service, List<String> added, List<String> removed*) There are situations in which a DeviceService will update the capabilities it supports and propagate these changes to the DeviceService. Such situations include:

- on discovery, DIALService will reach out to detect if certain apps are installed
- on discovery, certain DeviceServices need to reach out for version & region information

For more information on this particular method, see ConnectableDeviceDelegate's `connectableDevice:capabilitiesAdded:removed:` method.

Parameters:

- service – DeviceService that has experienced a change in capabilities
- added – List<String> of capabilities that are new to the DeviceService
- removed – List<String> of capabilities that the DeviceService has lost

void onDisconnect (*DeviceService service, Error error*) This method will be called on any disconnection. If error is nil, then the connection was clean and likely triggered by the responsible DiscoveryProvider or by the user.

Parameters:

- service – DeviceService that disconnected
- error – Error with a description of any errors causing the disconnect. If this value is nil, then the disconnect was clean/expected.

void onConnectionFailure (*DeviceService service, Error error*) Will be called if the DeviceService fails to establish a connection.

Parameters:

- service – DeviceService which has failed to connect
- error – Error with a description of the failure

void onPairingRequired (*DeviceService service, PairingType pairingType, Object pairingData*) If the DeviceService requires pairing, valuable data will be passed to the delegate via this method.

Parameters:

- service – DeviceService that requires pairing

- `pairingType` – `PairingType` that the `DeviceService` requires
- `pairingData` – Any data that might be required for the pairing process, will usually be `nil`

void onPairingSuccess (*DeviceService* `service`) **Parameters:**

- `service`

void onPairingFailed (*DeviceService* `service`, **Error** `error`) If there is any error in pairing, this method will be called.

Parameters:

- `service` – `DeviceService` that has failed to complete pairing
- `error` – `Error` with a description of the failure

FireTVService

`com.connectsdk.service.FireTVService`

extends `DeviceService`

`FireTVService` provides capabilities for FireTV devices. `FireTVService` acts as a layer on top of Fling SDK, and requires the Fling SDK library to function. `FireTVService` provides the following functionality:

- Media playback
- Media control

Using Connect SDK for discovery/control of FireTV devices will result in your app complying with the Fling SDK terms of service.

Properties

`final String ID = "FireTV"`

Inner Classes

- `ConvertResult`
- `PlayStateSubscription`
- `Subscription`

Methods

FireTVService (**ServiceDescription** `serviceDescription`, **ServiceConfig** `serviceConfig`) **Parameters:**

- `serviceDescription`
- `serviceConfig`

void connect () Prepare a service for usage

boolean isConnected () Check if service is ready

boolean isConnectable () Check if service implements connect/disconnect methods

void disconnect () Disconnect a service and close all subscriptions

CapabilityPriorityLevel **getPriorityLevel** (**Class**<?extends **CapabilityMethods** > *clazz*) Get a priority level for a particular capability

Parameters:

- *clazz*

MediaPlayer **getMediaPlayer** () Get MediaPlayer implementation

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel** () Get MediaPlayer priority level

void **getMediaInfo** (**final** **MediaInfoListener** *listener*) Get MediaInfo available only during playback otherwise returns an error

Parameters:

- *listener* – (optional) final MediaInfoListener with methods to be called on success or failure

ServiceSubscription <**MediaInfoListener**> **subscribeMediaInfo** (**MediaInfoListener** *listener*) Not supported

Parameters:

- *listener* – (optional) MediaInfoListener with methods to be called on success or failure

void **displayImage** (**String** *url*, **String** *mimeType*, **String** *title*, **String** *description*, **String** *iconSrc*, **final** **LaunchListener** *listener*) Display an image with metadata

Parameters:

- *url* – media source
- *mimeType*
- *title*
- *description*
- *iconSrc*
- *listener* – (optional) final LaunchListener with methods to be called on success or failure

void **playMedia** (**String** *url*, **String** *mimeType*, **String** *title*, **String** *description*, **String** *iconSrc*, **boolean** *shouldLoop*, **LaunchListener** *listener*) Play audio/video

Parameters:

- *url* – media source
- *mimeType*
- *title*
- *description*
- *iconSrc*
- *shouldLoop* – skipped in current implementation
- *listener* – (optional) LaunchListener with methods to be called on success or failure

void **closeMedia** (**LaunchSession** *launchSession*, **final** **ResponseListener** <**Object**> *listener*) Stop and close media player on FireTV. In current implementation it's similar to stop method

Parameters:

- *launchSession*
- *listener* – (optional) final ResponseListener< Object > with methods to be called on success or failure

void displayImage (*MediaInfo* *mediaInfo*, *LaunchListener* *listener*) Display an image with metadata

Parameters:

- *mediaInfo*
- *listener* – (optional) *LaunchListener* with methods to be called on success or failure

void playMedia (*MediaInfo* *mediaInfo*, *boolean* *shouldLoop*, *LaunchListener* *listener*) Play audio/video

Parameters:

- *mediaInfo*
- *shouldLoop* – skipped in current implementation
- *listener* – (optional) *LaunchListener* with methods to be called on success or failure

MediaControl **getMediaControl** () Get *MediaControl* capability. It should be used only during media playback.

CapabilityPriorityLevel **getMediaControlCapabilityLevel** () Get *MediaControl* priority level

void play (*ResponseListener* *<Object>* *listener*) Play current media.

Parameters:

- *listener* – (optional) *ResponseListener< Object >* with methods to be called on success or failure

void pause (*ResponseListener* *<Object>* *listener*) Pause current media.

Parameters:

- *listener* – (optional) *ResponseListener< Object >* with methods to be called on success or failure

void stop (*ResponseListener* *<Object>* *listener*) Stop current media and close FireTV application.

Parameters:

- *listener* – (optional) *ResponseListener< Object >* with methods to be called on success or failure

void rewind (*ResponseListener* *<Object>* *listener*) Not supported

Parameters:

- *listener* – (optional) *ResponseListener< Object >* with methods to be called on success or failure

void fastForward (*ResponseListener* *<Object>* *listener*) Not supported

Parameters:

- *listener* – (optional) *ResponseListener< Object >* with methods to be called on success or failure

void previous (*ResponseListener* *<Object>* *listener*) Not supported

Parameters:

- *listener* – (optional) *ResponseListener< Object >* with methods to be called on success or failure

void next (*ResponseListener* *<Object>* *listener*) Not supported

Parameters:

- *listener* – (optional) *ResponseListener< Object >* with methods to be called on success or failure

void seek (*long* *position*, *ResponseListener* *<Object>* *listener*) Seek current media.

Parameters:

- *position* – time in milliseconds
- *listener* – (optional) *ResponseListener< Object >* with methods to be called on success or failure

void getDuration (final *DurationListener* listener) Get current media duration.

Parameters:

- listener – (optional) final *DurationListener* with methods to be called on success or failure

void getPosition (final *PositionListener* listener) Get playback position

Parameters:

- listener – (optional) final *PositionListener* with methods to be called on success or failure

void getPlayState (final *PlayStateListener* listener) Get playback state

Parameters:

- listener – (optional) final *PlayStateListener* with methods to be called on success or failure

***ServiceSubscription* <*PlayStateListener*> subscribePlayState (final *PlayStateListener* listener)** Subscribe to playback state. Only single instance of subscription is available. Each new call returns the same subscription object.

Parameters:

- listener – (optional) final *PlayStateListener* with methods to be called on success or failure

static *DiscoveryFilter* discoveryFilter () Get filter instance for this service which contains a name of service and id. It is used in discovery process

Inherited Methods

void connect () Will attempt to connect to the *DeviceService*. The failure/success will be reported back to the *DeviceServiceListener*. If the connection attempt reveals that pairing is required, the *DeviceServiceListener* will also be notified in that event.

void disconnect () Will attempt to disconnect from the *DeviceService*. The failure/success will be reported back to the *DeviceServiceListener*.

boolean isConnected () Whether the *DeviceService* is currently connected

boolean isConnectable ()

void cancelPairing () Explicitly cancels pairing in services that require pairing. In some services, this will hide a prompt that is displaying on the device.

void sendPairingKey (String *pairingKey*) Will attempt to pair with the *DeviceService* with the provided pairing-Data. The failure/success will be reported back to the *DeviceServiceListener*.

Parameters:

- *pairingKey* – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

List<String> getCapabilities ()

boolean hasCapability (String *capability*) Test to see if the capabilities array contains a given capability. See the individual *Capability* classes for acceptable capability values.

It is possible to append a wildcard search term `.Any` to the end of the search term. This method will return true for capabilities that match the term up to the wildcard.

Example: `Launcher.App.Any`

Parameters:

- capability – Capability to test against

boolean hasAnyCapability (String... capabilities) Test to see if the capabilities array contains at least one capability in a given set of capabilities. See the individual Capability classes for acceptable capability values.

See hasCapability: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – Set of capabilities to test against

boolean hasCapabilities (List<String> capabilities) Test to see if the capabilities array contains a given set of capabilities. See the individual Capability classes for acceptable capability values.

See hasCapability: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – List of capabilities to test against

ServiceDescription **getServiceDescription ()**

ServiceConfig **getServiceConfig ()**

JSONObject **toJSONObject ()**

String getServiceName () Name of the DeviceService (webOS, Chromecast, etc)

void closeLaunchSession (LaunchSession launchSession, ResponseListener <Object> listener) Closes the session on the first screen device. Depending on the sessionType, the associated service will have different ways of handling the close functionality.

Parameters:

- launchSession – LaunchSession to close
- listener – (optional) listener to be called on success/failure

MediaPlayer **getMediaPlayer ()**

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel ()**

void getMediaInfo (MediaInfoListener listener) Parameters:

- listener – (optional) MediaInfoListener with methods to be called on success or failure

ServiceSubscription <MediaInfoListener> **subscribeMediaInfo (MediaInfoListener listener) Parameters:**

- listener – (optional) MediaInfoListener with methods to be called on success or failure

void displayImage (MediaInfo mediaInfo, LaunchListener listener) Display an image on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- MediaPlayer.Display.Image
- MediaPlayer.MediaData.Title
- MediaPlayer.MediaData.Description
- MediaPlayer.MediaData.Thumbnail
- MediaPlayer.MediaData.MimeType

Parameters:

- mediaInfo – Object of MediaInfo class which includes all the information about an image to display.
- listener – (optional) LaunchListener with methods to be called on success or failure

void playMedia (*MediaInfo mediaInfo*, *boolean shouldLoop*, *LaunchListener listener*) Play an audio or video file on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- `MediaPlayer.Play.Video`
- `MediaPlayer.Play.Audio`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- `mediaInfo` – Object of `MediaInfo` class which includes all the information about an image to display.
- `shouldLoop` – Whether to automatically loop playback
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void closeMedia (*LaunchSession launchSession*, *ResponseListener <Object> listener*) Close a running media session. Because media is handled differently on different platforms, it is required to keep track of `LaunchSession` and `MediaControl` objects to control that media session in the future. `LaunchSession` will be required to close the media and `mediaControl` will be required to control the media.

Related capabilities:

- `MediaPlayer.Close`

Parameters:

- `launchSession` – `LaunchSession` object for use in closing media instance
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

MediaControl `getMediaControl ()` Get `MediaControl` implementation

Returns: `MediaControl`

CapabilityPriorityLevel `getMediaControlCapabilityLevel ()` Get a capability priority for current implementation

Returns: `CapabilityPriorityLevel`

void play (*ResponseListener <Object> listener*) Send play command.

Related capabilities:

- `MediaControl.Play`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void pause (*ResponseListener <Object> listener*) Send pause command.

Related capabilities:

- `MediaControl.Pause`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void stop (*ResponseListener* <Object> *listener*) Send play command.

Related capabilities:

- `MediaControl.Stop`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void rewind (*ResponseListener* <Object> *listener*) Send rewind command.

Related capabilities:

- `MediaControl.Rewind`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void fastForward (*ResponseListener* <Object> *listener*) Send play command.

Related capabilities:

- `MediaControl.FastForward`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void previous (*ResponseListener* <Object> *listener*) This method is deprecated. Use `PlaylistControl::previous(ResponseListener<Object> listener)` instead.

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void next (*ResponseListener* <Object> *listener*) This method is deprecated. Use `PlaylistControl::next(ResponseListener<Object> listener)` instead.

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void seek (*long position*, *ResponseListener* <Object> *listener*) Seeks to a new position within the current media item

Related capabilities:

- `MediaControl.Seek`

Parameters:

- *position* – The new position, in milliseconds from the beginning of the stream
- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getDuration (*DurationListener listener*) Get the current media duration in milliseconds

Parameters:

- *listener* – (optional) `DurationListener` with methods to be called on success or failure

void getPosition (*PositionListener listener*) Get the current playback position in milliseconds

Parameters:

- *listener* – (optional) `PositionListener` with methods to be called on success or failure

void getPlayState (*PlayStateListener listener*) Get the current state of playback

Parameters:

- listener – (optional) PlayStateListener with methods to be called on success or failure

ServiceSubscription <PlayStateListener> **subscribePlayState** (*PlayStateListener listener*) Subscribe for playback state changes

Parameters:

- listener – receives play state notifications

Returns: ServiceSubscription<PlayStateListener>

void onLoseReachability (*DeviceServiceReachability reachability*) **Parameters:**

- reachability

void unsubscribe (*URLServiceSubscription<?> subscription*) **Parameters:**

- subscription

void sendCommand (*ServiceCommand<?> command*) **Parameters:**

- command

NetcastTVService

`com.connectsdk.service.NetcastTVService`

extends DeviceService <and-deviceservice>

NetcastTVService provides capabilities for LG Smart TVs running Netcast versions 3.x and 4.x (model years 2012-2014). The media playback functionality of NetcastTVService may be proxied through to DLNATVService to avoid requiring pairing. Commands & subscriptions on Netcast occur over HTTP.

The following capabilities are provided by the Netcast OS:

- Media playback
- Media control
- App launching*
- Volume control*
- Text input control*
- Key control (fiveway)*
- Mouse control*
- Power control*
- TV control (change channels, get channel info)*
- External input control*
- = requires pairing

To learn more about Netcast's second screen protocol, visit the [UDAP protocol specification](#).

Properties

```
final String ID = "Netcast TV"
final String UDAP_PATH_PAIRING = "/udap/api/pairing"
final String UDAP_PATH_DATA = "/udap/api/data"
final String UDAP_PATH_COMMAND = "/udap/api/command"
final String UDAP_PATH_EVENT = "/udap/api/event"
final String UDAP_PATH_APPTOAPP_DATA = "/udap/api/apptoapp/data/"
final String UDAP_PATH_APPTOAPP_COMMAND = "/udap/api/apptoapp/command/"
final String ROAP_PATH_APP_STORE = "/roap/api/command/"
final String UDAP_API_PAIRING = "pairing"
final String UDAP_API_COMMAND = "command"
final String UDAP_API_EVENT = "event"
final String TARGET_CHANNEL_LIST = "channel_list"
final String TARGET_CURRENT_CHANNEL = "cur_channel"
final String TARGET_VOLUME_INFO = "volume_info"
final String TARGET_APPLIST_GET = "applist_get"
final String TARGET_APPNUM_GET = "appnum_get"
final String TARGET_3D_MODE = "3DMode"
final String TARGET_IS_3D = "is_3D"
final String SMART_SHARE = "SmartShare?"
```

Inner Classes

- NetcastTVLaunchSessionR
- State

Methods

NetcastTVService (**ServiceDescription** *serviceDescription*, **ServiceConfig** *serviceConfig*) **Parameters:**

- serviceDescription
- serviceConfig

CapabilityPriorityLevel **getPriorityLevel** (**Class**<?extends **CapabilityMethods** > *clazz*) **Parameters:**

- clazz

void setServiceDescription (**ServiceDescription** *serviceDescription*) **Parameters:**

- serviceDescription

void **connect** ()

void **disconnect** ()

boolean **isConnectable** ()

boolean **isConnected** ()

void **onLoseReachability** (*DeviceServiceReachability reachability*) **Parameters:**

- reachability

void **hostByeBye** ()

void **showPairingKeyOnTV** ()

void **cancelPairing** ()

void **removePairingKeyOnTV** ()

void **sendPairingKey** (*final String pairingKey*) **Parameters:**

- pairingKey

Launcher **getLauncher** ()

CapabilityPriorityLevel **getLauncherCapabilityLevel** ()

void **getApplication** (*final String appName, final AppInfoListener listener*) **Parameters:**

- appName
- listener – (optional) final AppInfoListener with methods to be called on success or failure

void **launchApp** (*final String appId, final AppLaunchListener listener*) **Parameters:**

- appId
- listener – (optional) final AppLaunchListener with methods to be called on success or failure

void **launchAppWithInfo** (*AppInfo appInfo, Launcher.AppLaunchListener listener*) **Parameters:**

- appInfo
- listener – (optional) Launcher.AppLaunchListener with methods to be called on success or failure

void **launchAppWithInfo** (*AppInfo appInfo, Object params, Launcher.AppLaunchListener listener*) **Parameters:**

- appInfo
- params
- listener – (optional) Launcher.AppLaunchListener with methods to be called on success or failure

void **launchBrowser** (*String url, final Launcher.AppLaunchListener listener*) **Parameters:**

- url
- listener – (optional) final Launcher.AppLaunchListener with methods to be called on success or failure

void **launchYouTube** (*String contentId, Launcher.AppLaunchListener listener*) **Parameters:**

- contentId
- listener – (optional) Launcher.AppLaunchListener with methods to be called on success or failure

void **launchYouTube** (*final String contentId, float startTime, final AppLaunchListener listener*) **Parameters:**

- contentId

- startTime
- listener – (optional) final AppLaunchListener with methods to be called on success or failure

void launchHulu (final String *contentId*, final *Launcher.AppLaunchListener* listener) Parameters:

- contentId
- listener – (optional) final Launcher.AppLaunchListener with methods to be called on success or failure

void launchNetflix (final String *contentId*, final *Launcher.AppLaunchListener* listener) Parameters:

- contentId
- listener – (optional) final Launcher.AppLaunchListener with methods to be called on success or failure

void launchAppStore (final String *appId*, final *AppLaunchListener* listener) Parameters:

- appId
- listener – (optional) final AppLaunchListener with methods to be called on success or failure

void closeApp (*LaunchSession* launchSession, *ResponseListener* <Object> listener) Parameters:

- launchSession
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void getAppList (final *AppListListener* listener) Parameters:

- listener – (optional) final AppListListener with methods to be called on success or failure

void getRunningApp (*AppInfoListener* listener) Parameters:

- listener – (optional) AppInfoListener with methods to be called on success or failure

***ServiceSubscription* <*AppInfoListener*> subscribeRunningApp (*AppInfoListener* listener) Parameters:**

- listener – (optional) AppInfoListener with methods to be called on success or failure

void getAppState (final *LaunchSession* launchSession, final *AppStateListener* listener) Parameters:

- launchSession
- listener – (optional) final AppStateListener with methods to be called on success or failure

***ServiceSubscription* <*AppStateListener*> subscribeAppState (*LaunchSession* launchSession, *AppStateListener* listener) Parameters:**

- launchSession
- listener – (optional) AppStateListener with methods to be called on success or failure

***TVControl* getTVControl ()**

***CapabilityPriorityLevel* getTVControlCapabilityLevel ()**

void getChannelList (final *ChannelListListener* listener) Parameters:

- listener – (optional) final ChannelListListener with methods to be called on success or failure

void channelUp (*ResponseListener* <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void channelDown (*ResponseListener* <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void setChannel (final *ChannelInfo* channelInfo, final *ResponseListener* <Object> listener) Parameters:

- channelInfo
- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void getCurrentChannel (final *ChannelListener* listener) Parameters:

- listener – (optional) final ChannelListener with methods to be called on success or failure

***ServiceSubscription* <ChannelListener> subscribeCurrentChannel (final *ChannelListener* listener) Parameters:**

- listener – (optional) final ChannelListener with methods to be called on success or failure

void getProgramInfo (*ProgramInfoListener* listener) Parameters:

- listener – (optional) ProgramInfoListener with methods to be called on success or failure

***ServiceSubscription* <ProgramInfoListener> subscribeProgramInfo (*ProgramInfoListener* listener) Parameters:**

- listener – (optional) ProgramInfoListener with methods to be called on success or failure

void getProgramList (*ProgramListListener* listener) Parameters:

- listener – (optional) ProgramListListener with methods to be called on success or failure

***ServiceSubscription* <ProgramListListener> subscribeProgramList (*ProgramListListener* listener) Parameters:**

- listener – (optional) ProgramListListener with methods to be called on success or failure

void set3DEnabled (final boolean enabled, final *ResponseListener* <Object> listener) Parameters:

- enabled
- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void get3DEnabled (final *State3DModeListener* listener) Parameters:

- listener – (optional) final State3DModeListener with methods to be called on success or failure

***ServiceSubscription* <State3DModeListener> subscribe3DEnabled (final *State3DModeListener* listener) Parameters:**

- listener – (optional) final State3DModeListener with methods to be called on success or failure

***VolumeControl* getVolumeControl ()**

***CapabilityPriorityLevel* getVolumeControlCapabilityLevel ()**

void volumeUp (*ResponseListener* <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void volumeDown (*ResponseListener* <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void setVolume (float volume, *ResponseListener* <Object> listener) Parameters:

- volume
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void getVolume (final *VolumeListener* listener) Parameters:

- listener – (optional) final VolumeListener with methods to be called on success or failure

void setMute (final boolean isMute, final *ResponseListener* <Object> listener) Parameters:

- isMute

- listener – (optional) final `ResponseListener< Object >` with methods to be called on success or failure

void getMute (final *MuteListener* listener) Parameters:

- listener – (optional) final `MuteListener` with methods to be called on success or failure

ServiceSubscription <VolumeListener> **subscribeVolume (*VolumeListener* listener) Parameters:**

- listener – (optional) `VolumeListener` with methods to be called on success or failure

ServiceSubscription <MuteListener> **subscribeMute (*MuteListener* listener) Parameters:**

- listener – (optional) `MuteListener` with methods to be called on success or failure

ExternalInputControl **getExternalInput ()**

CapabilityPriorityLevel **getExternalInputControlPriorityLevel ()**

void launchInputPicker (final *AppLaunchListener* listener) Parameters:

- listener – (optional) final `AppLaunchListener` with methods to be called on success or failure

void closeInputPicker (*LaunchSession* launchSession, *ResponseListener <Object>* listener) Parameters:

- launchSession
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getExternalInputList (*ExternalInputListListener* listener) Parameters:

- listener – (optional) `ExternalInputListListener` with methods to be called on success or failure

void setExternalInput (*ExternalInputInfo* input, *ResponseListener <Object>* listener) Parameters:

- input
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

MediaPlayer **getMediaPlayer ()**

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel ()**

void getMediaInfo (final *MediaInfoListener* listener) Parameters:

- listener – (optional) final `MediaInfoListener` with methods to be called on success or failure

ServiceSubscription <MediaInfoListener> **subscribeMediaInfo (*MediaInfoListener* listener) Parameters:**

- listener – (optional) `MediaInfoListener` with methods to be called on success or failure

void displayImage (final String url, final String mimeType, final String title, final String description, final String iconSrc, final *MediaPlayer.LaunchListener* listener) Parameters:

- url
- mimeType
- title
- description
- iconSrc
- listener – (optional) final `MediaPlayer.LaunchListener` with methods to be called on success or failure

void displayImage (*MediaInfo* mediaInfo, *LaunchListener* listener) Parameters:

- mediaInfo
- listener – (optional) `LaunchListener` with methods to be called on success or failure

void playMedia (String url, String mimeType, String title, String description, String iconSrc, boolean shouldLoop, MediaPlayer.LaunchListener listener)
Parameters:

- url
- mimeType
- title
- description
- iconSrc
- shouldLoop
- listener – (optional) MediaPlayer.LaunchListener with methods to be called on success or failure

void playMedia (MediaInfo mediaInfo, boolean shouldLoop, final MediaPlayer.LaunchListener listener)
Parameters:

- mediaInfo
- shouldLoop
- listener – (optional) final MediaPlayer.LaunchListener with methods to be called on success or failure

void closeMedia (LaunchSession launchSession, ResponseListener <Object> listener) Parameters:

- launchSession
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

MediaControl getMediaControl () Get MediaControl implementation

Returns: MediaControl

CapabilityPriorityLevel getMediaControlCapabilityLevel () Get a capability priority for current implementation

Returns: CapabilityPriorityLevel

void play (ResponseListener <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void pause (ResponseListener <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void stop (final ResponseListener <Object> listener) Parameters:

- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void rewind (ResponseListener <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void fastForward (ResponseListener <Object> listener) Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void previous (ResponseListener <Object> listener) This method is deprecated. Use `PlaylistControl::previous (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void next (*ResponseListener* <Object> *listener*) This method is deprecated. Use `PlaylistControl::next (ResponseListener<Object> listener)` instead.

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void seek (long *position*, *ResponseListener* <Object> *listener*) **Parameters:**

- *position* – The new position, in milliseconds from the beginning of the stream
- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getDuration (*DurationListener* *listener*) Get the current media duration in milliseconds

Parameters:

- *listener* – (optional) `DurationListener` with methods to be called on success or failure

void getPosition (*PositionListener* *listener*) Get the current playback position in milliseconds

Parameters:

- *listener* – (optional) `PositionListener` with methods to be called on success or failure

void getPlayState (*PlayStateListener* *listener*) Get the current state of playback

Parameters:

- *listener* – (optional) `PlayStateListener` with methods to be called on success or failure

ServiceSubscription <*PlayStateListener*> **subscribePlayState** (*PlayStateListener* *listener*) Subscribe for playback state changes

Parameters:

- *listener* – receives play state notifications

Returns: `ServiceSubscription<PlayStateListener>`

MouseControl **getMouseControl** ()

CapabilityPriorityLevel **getMouseControlCapabilityLevel** ()

void connectMouse ()

void disconnectMouse ()

void click ()

void move (double *dx*, double *dy*) **Parameters:**

- *dx*
- *dy*

void move (*PointF* *diff*) **Parameters:**

- *diff*

void scroll (double *dx*, double *dy*) **Parameters:**

- *dx*
- *dy*

void scroll (*PointF* *diff*) **Parameters:**

- *diff*

TextInputControl **getTextInputControl** ()

CapabilityPriorityLevel **getTextInputControlCapabilityLevel** ()

ServiceSubscription **<TextInputStatusListener> subscribeTextInputStatus** (final *TextInputStatusListener* listener)

Parameters:

- listener – (optional) final *TextInputStatusListener* with methods to be called on success or failure

void sendText (final *String* input) **Parameters:**

- input

void sendEnter ()

void sendDelete ()

KeyControl **getKeyControl** ()

CapabilityPriorityLevel **getKeyControlCapabilityLevel** ()

void up (final *ResponseListener* **<Object>** listener) **Parameters:**

- listener – (optional) final *ResponseListener***< Object >** with methods to be called on success or failure

void down (final *ResponseListener* **<Object>** listener) **Parameters:**

- listener – (optional) final *ResponseListener***< Object >** with methods to be called on success or failure

void left (final *ResponseListener* **<Object>** listener) **Parameters:**

- listener – (optional) final *ResponseListener***< Object >** with methods to be called on success or failure

void right (final *ResponseListener* **<Object>** listener) **Parameters:**

- listener – (optional) final *ResponseListener***< Object >** with methods to be called on success or failure

void ok (final *ResponseListener* **<Object>** listener) **Parameters:**

- listener – (optional) final *ResponseListener***< Object >** with methods to be called on success or failure

void back (final *ResponseListener* **<Object>** listener) **Parameters:**

- listener – (optional) final *ResponseListener***< Object >** with methods to be called on success or failure

void home (final *ResponseListener* **<Object>** listener) **Parameters:**

- listener – (optional) final *ResponseListener***< Object >** with methods to be called on success or failure

PowerControl **getPowerControl** ()

CapabilityPriorityLevel **getPowerControlCapabilityLevel** ()

void powerOff (*ResponseListener* **<Object>** listener) **Parameters:**

- listener – (optional) *ResponseListener***< Object >** with methods to be called on success or failure

void powerOn (*ResponseListener* **<Object>** listener) **Parameters:**

- listener – (optional) *ResponseListener***< Object >** with methods to be called on success or failure

String getHttpMessageForHandleKeyInput (final *int* keycode) **Parameters:**

- keycode

void sendKeyCode (*KeyCode* keycode, *ResponseListener* **<Object>** listener) **Parameters:**

- keycode
- listener – (optional) *ResponseListener***< Object >** with methods to be called on success or failure

String decToHex (String *dec*) Parameters:

- *dec*

String decToHex (long *dec*) Parameters:

- *dec*

void sendCommand (final ServiceCommand<?> *mCommand*) Parameters:

- *mCommand*

void unsubscribe (URLServiceSubscription<?> *subscription*) Parameters:

- *subscription*

DLNATService **getDLNATService ()**

DIALService **getDIALService ()**

static DiscoveryFilter **discoveryFilter ()**

Inherited Methods

void connect () Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceListener. If the connection attempt reveals that pairing is required, the DeviceServiceListener will also be notified in that event.

void disconnect () Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceListener.

boolean isConnected () Whether the DeviceService is currently connected

boolean isConnectable ()

void cancelPairing () Explicitly cancels pairing in services that require pairing. In some services, this will hide a prompt that is displaying on the device.

void sendPairingKey (String *pairingKey*) Will attempt to pair with the DeviceService with the provided pairing-Data. The failure/success will be reported back to the DeviceServiceListener.

Parameters:

- *pairingKey* – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

List<String> **getCapabilities ()**

boolean hasCapability (String *capability*) Test to see if the capabilities array contains a given capability. See the individual Capability classes for acceptable capability values.

It is possible to append a wildcard search term `.Any` to the end of the search term. This method will return true for capabilities that match the term up to the wildcard.

Example: `Launcher.App.Any`

Parameters:

- *capability* – Capability to test against

boolean hasAnyCapability (String... *capabilities*) Test to see if the capabilities array contains at least one capability in a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – Set of capabilities to test against

boolean hasCapabilities (List<String> capabilities) Test to see if the capabilities array contains a given set of capabilities. See the individual Capability classes for acceptable capability values.

See hasCapability: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – List of capabilities to test against

ServiceDescription **getServiceDescription ()**

ServiceConfig **getServiceConfig ()**

JSONObject **toJSONObject ()**

String getServiceName () Name of the DeviceService (webOS, Chromecast, etc)

void closeLaunchSession (LaunchSession launchSession, ResponseListener <Object> listener) Closes the session on the first screen device. Depending on the sessionType, the associated service will have different ways of handling the close functionality.

Parameters:

- launchSession – LaunchSession to close
- listener – (optional) listener to be called on success/failure

Launcher **getLauncher ()**

CapabilityPriorityLevel **getLauncherCapabilityLevel ()**

void launchAppWithInfo (AppInfo appInfo, AppLaunchListener listener) Launch an application on the device.

Related capabilities:

- Launcher.App
- Launcher.App.Params – if launching with params

Parameters:

- appInfo – AppInfo object for the application
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void launchApp (String appId, AppLaunchListener listener) Launch an application on the device.

Related capabilities:

- Launcher.App

Parameters:

- appId – ID of the application
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void closeApp (LaunchSession launchSession, ResponseListener <Object> listener) Close an application on the device.

Related capabilities:

- Launcher.App.Close

Parameters:

- launchSession – LaunchSession of the target app

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getAppList (*AppListListener listener*) Gets a list of all apps installed on the device.

Related capabilities:

- `Launcher.App.List`

Parameters:

- listener – (optional) `AppListListener` with methods to be called on success or failure

void getRunningApp (*AppInfoListener listener*) Gets an `AppInfo` object for the current running app on the device.

Related capabilities:

- `Launcher.RunningApp`

Parameters:

- listener – (optional) `AppInfoListener` with methods to be called on success or failure

ServiceSubscription <AppInfoListener> **subscribeRunningApp** (*AppInfoListener listener*) Subscribes to changes of the current running app. Every time the running app changes, the success block will be called with an `AppInfo` object for the current running app.

Related capabilities:

- `Launcher.RunningApp.Subscribe`

Parameters:

- listener – (optional) `AppInfoListener` with methods to be called on success or failure

void getAppState (*LaunchSession launchSession, AppStateListener listener*) Gets the target app's running status and on-screen visibility.

Related capabilities:

- `Launcher.AppState`

Parameters:

- `launchSession` – `LaunchSession` of the target app
- listener – (optional) `AppStateListener` with methods to be called on success or failure

ServiceSubscription <AppStateListener> **subscribeAppState** (*LaunchSession launchSession, AppStateListener listener*) Subscribes to changes of the state of the target app. Every time the app's state changes, the success block will be called with info on the app's running status and on-screen visibility.

Related capabilities:

- `Launcher.AppState.Subscribe`

Parameters:

- `launchSession` – `LaunchSession` of the target app
- listener – (optional) `AppStateListener` with methods to be called on success or failure

void launchBrowser (*String url, AppLaunchListener listener*) Launch the web browser. Will launch deep-linked to provided URL, if supported on the target platform.

Related capabilities:

- `Launcher.Browser`
- `Launcher.Browser.Params` – if launching with url

Parameters:

- url
- listener – (optional) `AppLaunchListener` with methods to be called on success or failure

void launchYouTube (String *contentId*, *AppLaunchListener* listener) Launch YouTube app. Will launch deep-linked to provided *contentId*, if supported on the target platform.

Related capabilities:

- `Launcher.YouTube`
- `Launcher.YouTube.Params` – if launching with *contentId*

Parameters:

- *contentId* – Video id to open
- listener – (optional) `AppLaunchListener` with methods to be called on success or failure

void launchNetflix (String *contentId*, *AppLaunchListener* listener) Launch Netflix app. Will launch deep-linked to provided *contentId*, if supported on the target platform.

Related capabilities:

- `Launcher.Netflix`
- `Launcher.Netflix.Params` – if launching with *contentId*

Parameters:

- *contentId* – Video id to open
- listener – (optional) `AppLaunchListener` with methods to be called on success or failure

void launchHulu (String *contentId*, *AppLaunchListener* listener) Launch Hulu app. Will launch deep-linked to provided *contentId*, if supported on the target platform.

Related capabilities:

- `Launcher.Hulu`
- `Launcher.Hulu.Params` – if launching with *contentId*

Parameters:

- *contentId* – Video id to open
- listener – (optional) `AppLaunchListener` with methods to be called on success or failure

void launchAppStore (String *appId*, *AppLaunchListener* listener) Launch the device's app store app, optionally deep-linked to a specific app's page.

Related capabilities:

- `Launcher.AppStore`
- `Launcher.AppStore.Params`

Parameters:

- *appId* – (optional) ID of the application to show in the app store
- listener – (optional) `AppLaunchListener` with methods to be called on success or failure

***MediaControl* getMediaControl ()** Get `MediaControl` implementation

Returns: `MediaControl`

CapabilityPriorityLevel **getMediaControlCapabilityLevel ()** Get a capability priority for current implementation

Returns: CapabilityPriorityLevel

void play (*ResponseListener* <Object> *listener*) Send play command.

Related capabilities:

- `MediaControl.Play`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void pause (*ResponseListener* <Object> *listener*) Send pause command.

Related capabilities:

- `MediaControl.Pause`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void stop (*ResponseListener* <Object> *listener*) Send play command.

Related capabilities:

- `MediaControl.Stop`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void rewind (*ResponseListener* <Object> *listener*) Send rewind command.

Related capabilities:

- `MediaControl.Rewind`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void fastForward (*ResponseListener* <Object> *listener*) Send play command.

Related capabilities:

- `MediaControl.FastForward`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void previous (*ResponseListener* <Object> *listener*) This method is deprecated. Use `PlaylistControl::previous (ResponseListener<Object> listener)` instead.

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void next (*ResponseListener* <Object> *listener*) This method is deprecated. Use `PlaylistControl::next (ResponseListener<Object> listener)` instead.

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void seek (*long position*, *ResponseListener* <Object> *listener*) Seeks to a new position within the current media item

Related capabilities:

- `MediaControl.Seek`

Parameters:

- *position* – The new position, in milliseconds from the beginning of the stream
- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getDuration (*DurationListener listener*) Get the current media duration in milliseconds

Parameters:

- *listener* – (optional) `DurationListener` with methods to be called on success or failure

void getPosition (*PositionListener listener*) Get the current playback position in milliseconds

Parameters:

- *listener* – (optional) `PositionListener` with methods to be called on success or failure

void getPlayState (*PlayStateListener listener*) Get the current state of playback

Parameters:

- *listener* – (optional) `PlayStateListener` with methods to be called on success or failure

ServiceSubscription <*PlayStateListener*> **subscribePlayState** (*PlayStateListener listener*) Subscribe for playback state changes

Parameters:

- *listener* – receives play state notifications

Returns: `ServiceSubscription<PlayStateListener>`

MediaPlayer **getMediaPlayer** ()

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel** ()

void getMediaInfo (*MediaInfoListener listener*) **Parameters:**

- *listener* – (optional) `MediaInfoListener` with methods to be called on success or failure

ServiceSubscription <*MediaInfoListener*> **subscribeMediaInfo** (*MediaInfoListener listener*) **Parameters:**

- *listener* – (optional) `MediaInfoListener` with methods to be called on success or failure

void displayImage (*MediaInfo mediaInfo*, *LaunchListener listener*) Display an image on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- `MediaPlayer.Display.Image`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- *mediaInfo* – Object of `MediaInfo` class which includes all the information about an image to display.

- listener – (optional) `LaunchListener` with methods to be called on success or failure

void playMedia (*MediaInfo* `mediaInfo`, *boolean* `shouldLoop`, *LaunchListener* `listener`) Play an audio or video file on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- `MediaPlayer.Play.Video`
- `MediaPlayer.Play.Audio`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- `mediaInfo` – Object of `MediaInfo` class which includes all the information about an image to display.
- `shouldLoop` – Whether to automatically loop playback
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void closeMedia (*LaunchSession* `launchSession`, *ResponseListener* `<Object> listener`) Close a running media session. Because media is handled differently on different platforms, it is required to keep track of `LaunchSession` and `MediaControl` objects to control that media session in the future. `LaunchSession` will be required to close the media and `mediaControl` will be required to control the media.

Related capabilities:

- `MediaPlayer.Close`

Parameters:

- `launchSession` – `LaunchSession` object for use in closing media instance
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

TVControl **getTVControl** ()

CapabilityPriorityLevel **getTVControlCapabilityLevel** ()

void channelUp (*ResponseListener* `<Object> listener`) Sends a channel up command to the TV.

Related capabilities:

- `TVControl.Channel.Up`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void channelDown (*ResponseListener* `<Object> listener`) Sends a channel down command to the TV.

Related capabilities:

- `TVControl.Channel.Down`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void setChannel (*ChannelInfo* `channelNumber`, *ResponseListener* `<Object> listener`) Sets the current channel to the channel provided by the `ChannelInfo` object provided.

Related capabilities:

- `TVControl.Channel.Set`

Parameters:

- `channelNumber`
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void `getCurrentChannel` (*`ChannelListener listener`*) Gets the current channel info from the TV.

Related capabilities:

- `TVControl.Channel.Get`

Parameters:

- `listener` – (optional) `ChannelListener` with methods to be called on success or failure

`ServiceSubscription <ChannelListener>` **subscribeCurrentChannel** (*`ChannelListener listener`*) Subscribes to any changes in the current channel. Each time the channel is changed, the new channel's info will be provided to the success callback.

Related capabilities:

- `TVControl.Channel.Subscribe`

Parameters:

- `listener` – (optional) `ChannelListener` with methods to be called on success or failure

void `getChannelList` (*`ChannelListListener listener`*) Get a list of available channels from the TV.

Related capabilities:

- `TVControl.Channel.List`

Parameters:

- `listener` – (optional) `ChannelListListener` with methods to be called on success or failure

void `getProgramInfo` (*`ProgramInfoListener listener`*) Gets the current program info from the TV.

Related capabilities:

- `TVControl.Program.Get`

Parameters:

- `listener` – (optional) `ProgramInfoListener` with methods to be called on success or failure

`ServiceSubscription <ProgramInfoListener>` **subscribeProgramInfo** (*`ProgramInfoListener listener`*) Subscribes to any changes in the current program. Each time the channel is changed or a new program starts, the new program's info will be provided to the success callback.

Related capabilities:

- `TVControl.Program.Subscribe`

Parameters:

- `listener` – (optional) `ProgramInfoListener` with methods to be called on success or failure

void `getProgramList` (*`ProgramListListener listener`*) Gets a list of all programs scheduled to play on the current channel.

Related capabilities:

- `TVControl.Program.List`

Parameters:

- listener – (optional) ProgramListListener with methods to be called on success or failure

ServiceSubscription <ProgramListListener> **subscribeProgramList** (*ProgramListListener* listener) Subscribes to any changes in the current program. Each time the channel is changed or a new program starts, the new program's info will be provided to the success callback.

Related capabilities:

- TVControl.Program.List.Subscribe

Parameters:

- listener – (optional) ProgramListListener with methods to be called on success or failure

void get3DEnabled (*State3DModeListener* listener) Gets the current 3D status of the TV.

Related capabilities:

- TVControl.3D.Get

Parameters:

- listener – (optional) State3DModeListener with methods to be called on success or failure

void set3DEnabled (boolean enabled, *ResponseListener* <Object> listener) Sets the current 3D status of the TV.

Related capabilities:

- TVControl.3D.Set

Parameters:

- enabled – Whether the TV's 3D mode should be on or off
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

ServiceSubscription <State3DModeListener> **subscribe3DEnabled** (*State3DModeListener* listener) Subscribes to changes in the TV's 3D status.

Related capabilities:

- TVControl.3D.Subscribe

Parameters:

- listener – (optional) State3DModeListener with methods to be called on success or failure

VolumeControl **getVolumeControl** ()

CapabilityPriorityLevel **getVolumeControlCapabilityLevel** ()

void volumeUp (*ResponseListener* <Object> listener) Sends the volume up command to the device.

Related capabilities:

- VolumeControl.UpDown

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void volumeDown (*ResponseListener* <Object> listener) Sends the volume down command to the device.

Related capabilities:

- VolumeControl.UpDown

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void setVolume (float volume, *ResponseListener* <Object> listener) Set the volume of the device.

Related capabilities:

- `VolumeControl.Set`

Parameters:

- volume – Volume as a float between 0.0 and 1.0
- listener – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void getVolume (*VolumeListener* listener) Get the current volume of the device.

Related capabilities:

- `VolumeControl.Get`

Parameters:

- listener – (optional) *VolumeListener* with methods to be called on success or failure

void setMute (boolean isMute, *ResponseListener* <Object> listener) Set the current volume.

Related capabilities:

- `VolumeControl.Mute.Set`

Parameters:

- isMute
- listener – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void getMute (*MuteListener* listener) Get the current mute state.

Related capabilities:

- `VolumeControl.Mute.Get`

Parameters:

- listener – (optional) *MuteListener* with methods to be called on success or failure

ServiceSubscription <*VolumeListener*> **subscribeVolume (*VolumeListener* listener)** Subscribe to the volume on the TV.

Related capabilities:

- `VolumeControl.Subscribe`

Parameters:

- listener – (optional) *VolumeListener* with methods to be called on success or failure

ServiceSubscription <*MuteListener*> **subscribeMute (*MuteListener* listener)** Subscribe to the mute state on the TV.

Related capabilities:

- `VolumeControl.Mute.Subscribe`

Parameters:

- listener – (optional) *MuteListener* with methods to be called on success or failure

ExternalInputControl **getExternalInput ()**

CapabilityPriorityLevel **getExternalInputControlPriorityLevel ()**

void launchInputPicker (*AppLaunchListener listener*) Launches the visual input picker on the device. This may be helpful for situations where the device does not support directly listing/modifying the external inputs.

Related capabilities:

- `ExternalInputControl.Picker.Launch`

Parameters:

- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void closeInputPicker (*LaunchSession launchSessionm, ResponseListener <Object> listener*) Closes the input picker on the device, if it is currently open.

Related capabilities:

- `ExternalInputControl.Picker.Close`

Parameters:

- `launchSessionm`
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getExternalInputList (*ExternalInputListListener listener*) Get a list of input devices (HDMI, AV, etc) connected to the device

Related capabilities:

- `ExternalInputControl.List`

Parameters:

- `listener` – (optional) `ExternalInputListListener` with methods to be called on success or failure

void setExternalInput (*ExternalInputInfo input, ResponseListener <Object> listener*) Switch to the specified external input

Related capabilities:

- `ExternalInputControl.Set`

Parameters:

- `input`
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

MouseControl **getMouseControl** ()

CapabilityPriorityLevel **getMouseControlCapabilityLevel** ()

void connectMouse () Establish a connection with the DeviceService's mouse communication medium (WebSocket, HTTP, etc). While this step may not be necessary with certain platforms, it is suggested to call it anyways, for purposes of seamless normalization. Calling connect on a non-connectable protocol will just trigger the success callback immediately.

Related capabilities:

- `MouseControl.Connect`

void disconnectMouse () Disconnects from the mouse communication medium.

Related capabilities:

- `MouseControl.Disconnect`

void click () Perform a click action at the current mouse position.

Related capabilities:

- `MouseControl.Click`

void move (double *dx*, double *dy*) Move the mouse by the given distance values.

Related capabilities:

- `MouseControl.Move`

Parameters:

- *dx* – Distance to move the mouse on the x-axis relative to its current position
- *dy* – Distance to move the mouse on the y-axis relative to its current position

void scroll (double *dx*, double *dy*) Scroll by the given distance values.

Related capabilities:

- `MouseControl.Scroll`

Parameters:

- *dx* – Distance to scroll the mouse on the x-axis relative to its current position
- *dy* – Distance to scroll the mouse on the y-axis relative to its current position

TextInputControl **getTextInputControl ()**

CapabilityPriorityLevel **getTextInputControlCapabilityLevel ()**

ServiceSubscription **<TextInputStatusListener> subscribeTextInputStatus (*TextInputStatusListener* listener)**

Subscribe to information about the current text field.

Related capabilities:

- `TextInputControl.Subscribe`

Parameters:

- *listener* – (optional) `TextInputStatusListener` with methods to be called on success or failure

void sendText (String *input*) Send text to the current text field.

Related capabilities:

- `TextInputControl.Send.Text`

Parameters:

- *input*

void sendEnter () Send enter key to the current text field.

Related capabilities:

- `TextInputControl.Send.Enter`

void sendDelete () Send delete event to the current text field.

Related capabilities:

- `TextInputControl.Send.Delete`

PowerControl **getPowerControl ()**

CapabilityPriorityLevel **getPowerControlCapabilityLevel ()**

void powerOff (*ResponseListener* <Object> *listener*) Sends a power off signal to the TV. A success message will, internally, trigger a disconnection with the device.

Related capabilities:

- `PowerControl.Off`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void powerOn (*ResponseListener* <Object> *listener*)

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

KeyControl **getKeyControl** ()

CapabilityPriorityLevel **getKeyControlCapabilityLevel** ()

void up (*ResponseListener* <Object> *listener*) Sends the up button key code to the TV.

Related capabilities:

- `KeyControl.Up`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void down (*ResponseListener* <Object> *listener*) Sends the down button key code to the TV.

Related capabilities:

- `KeyControl.Down`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void left (*ResponseListener* <Object> *listener*) Sends the left button key code to the TV.

Related capabilities:

- `KeyControl.Left`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void right (*ResponseListener* <Object> *listener*) Sends the right button key code to the TV.

Related capabilities:

- `KeyControl.Right`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void ok (*ResponseListener* <Object> *listener*) Sends the OK button key code to the TV.

Related capabilities:

- `KeyControl.OK`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void back (*ResponseListener* <Object> *listener*) Sends the back button key code to the TV.

Related capabilities:

- `KeyControl.Back`

Parameters:

- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void home (*ResponseListener* <Object> *listener*) Sends the home button key code to the TV.

Related capabilities:

- `KeyControl.Home`

Parameters:

- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void sendKeyCode (*KeyCode* *keycode*, *ResponseListener* <Object> *listener*) Sends a key code value to the TV.

Related capabilities:

- `KeyControl.Send.KeyCode`

Parameters:

- *keycode*
- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void onLoseReachability (*DeviceServiceReachability* *reachability*) **Parameters:**

- *reachability*

void unsubscribe (*URLServiceSubscription*<?> *subscription*) **Parameters:**

- *subscription*

void sendCommand (*ServiceCommand*<?> *command*) **Parameters:**

- *command*

RokuService

`com.connectsdk.service.RokuService`

extends *DeviceService*

RokuService provides many capabilities for Roku devices. Communication with Roku devices occurs over HTTP.

- List, launch, & close apps
- Media playback
- Media control
- Text input control
- Key control (fiveway)

These APIs should work on all Roku devices – they have been tested on Roku 2, Roku 3, and Roku Streaming Stick all running Roku 5.3 or later.

To learn more about the Roku External Control APIs, visit the [Roku External Control Guide](#).

Properties

final String ID = "Roku"

Inner Classes

- RokuLaunchSession

Methods

static void **registerApp** (String *appId*)

Parameters:

- *appId*

static DiscoveryFilter **discoveryFilter** ()

RokuService (ServiceDescription *serviceDescription*, ServiceConfig *serviceConfig*)

Parameters:

- *serviceDescription*
- *serviceConfig*

void **setServiceDescription** (ServiceDescription *serviceDescription*)

Parameters:

- *serviceDescription*

CapabilityPriorityLevel **getPriorityLevel** (Class<?extends CapabilityMethods > *clazz*)

Parameters:

- *clazz*

Launcher **getLauncher** ()

CapabilityPriorityLevel **getLauncherCapabilityLevel** ()

void **launchApp** (String *appId*, *AppLaunchListener* *listener*)

Parameters:

- *appId*
- *listener* – (optional) *AppLaunchListener* with methods to be called on success or failure

void **launchAppWithInfo** (*AppInfo* *appInfo*, *Launcher*. *AppLaunchListener* *listener*)

Parameters:

- *appInfo*
- *listener* – (optional) *Launcher.AppLaunchListener* with methods to be called on success or failure

void **launchAppWithInfo** (final *AppInfo* *appInfo* , Object *params*, final *Launcher*. *AppLaunchListener* *listener*)

Parameters:

- *appInfo*

- params
- listener – (optional) final Launcher.AppLaunchListener with methods to be called on success or failure

void **closeApp** (*LaunchSession* launchSession, *ResponseListener* <Object> listener)

Parameters:

- launchSession
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **getAppList** (final *AppListListener* listener)

Parameters:

- listener – (optional) final AppListListener with methods to be called on success or failure

void **getRunningApp** (*AppInfoListener* listener)

Parameters:

- listener – (optional) AppInfoListener with methods to be called on success or failure

ServiceSubscription < *AppInfoListener* > **subscribeRunningApp** (*AppInfoListener* listener)

Parameters:

- listener – (optional) AppInfoListener with methods to be called on success or failure

void **getAppState** (*LaunchSession* launchSession, *AppStateListener* listener)

Parameters:

- listener – (optional) AppStateListener with methods to be called on success or failure

ServiceSubscription < *AppStateListener* > **subscribeAppState** (*LaunchSession* launchSession, *AppStateListener* listener)

Parameters:

- launchSession
- listener – (optional) AppStateListener with methods to be called on success or failure

void **launchBrowser** (String url, *Launcher.AppLaunchListener* listener)

Parameters:

- url
- listener – (optional) Launcher.AppLaunchListener with methods to be called on success or failure

void **launchYouTube** (String contentId, *Launcher.AppLaunchListener* listener)

Parameters:

- contentId
- listener – (optional) Launcher.AppLaunchListener with methods to be called on success or failure

void **launchYouTube** (String contentId, float startTime, *AppLaunchListener* listener)

Parameters:

- contentId

- startTime
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void **launchNetflix** (final String *contentId*, final *Launcher.AppLaunchListener* listener)

Parameters:

- contentId
- listener – (optional) final Launcher.AppLaunchListener with methods to be called on success or failure

void **launchHulu** (final String *contentId*, final *Launcher.AppLaunchListener* listener)

Parameters:

- contentId
- listener – (optional) final Launcher.AppLaunchListener with methods to be called on success or failure

void **launchAppStore** (final String *appId*, *AppLaunchListener* listener)

Parameters:

- appId
- listener – (optional) AppLaunchListener with methods to be called on success or failure

KeyControl **getKeyControl** ()

CapabilityPriorityLevel **getKeyControlCapabilityLevel** ()

void **up** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **down** (final *ResponseListener* <Object> listener)

Parameters:

- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void **left** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **right** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **ok** (final *ResponseListener* <Object> listener)

Parameters:

- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void **back** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **home** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

MediaControl **getMediaControl** ()

Get `MediaControl` implementation

Returns: `MediaControl`

CapabilityPriorityLevel **getMediaControlCapabilityLevel** ()

Get a capability priority for current implementation

Returns: `CapabilityPriorityLevel`

void **play** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **pause** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **stop** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **rewind** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **fastForward** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **previous** (*ResponseListener* <Object> listener)

This method is deprecated. Use `PlaylistControl::previous (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **next** (*ResponseListener* <Object> listener)

This method is deprecated. Use `PlaylistControl::next (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **getDuration** (*DurationListener* listener)

Get the current media duration in milliseconds

Parameters:

- listener – (optional) DurationListener with methods to be called on success or failure

void **getPosition** (*PositionListener* listener)

Get the current playback position in milliseconds

Parameters:

- listener – (optional) PositionListener with methods to be called on success or failure

void **seek** (long position, *ResponseListener* <Object> listener)

Parameters:

- position – The new position, in milliseconds from the beginning of the stream
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

MediaPlayer **getMediaPlayer** ()

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel** ()

void **getMediaInfo** (*MediaInfoListener* listener)

Parameters:

- listener – (optional) MediaInfoListener with methods to be called on success or failure

ServiceSubscription <*MediaInfoListener*> **subscribeMediaInfo** (*MediaInfoListener* listener)

Parameters:

- listener – (optional) MediaInfoListener with methods to be called on success or failure

void **displayImage** (String url, String mimeType, String title, String description, String iconSrc, *MediaPlayer.LaunchListener* listener)

Parameters:

- url
- mimeType
- title
- description
- iconSrc
- listener – (optional) MediaPlayer.LaunchListener with methods to be called on success or failure

void **displayImage** (*MediaInfo* mediaInfo, *MediaPlayer.LaunchListener* listener)

Parameters:

- mediaInfo
- listener – (optional) MediaPlayer.LaunchListener with methods to be called on success or failure

void **playMedia** (String url, String mimeType, String title, String description, String iconSrc, boolean shouldLoop, *MediaPlayer.LaunchListener* listener)

Parameters:

- url

- mimeType
- title
- description
- iconSrc
- shouldLoop
- listener – (optional) MediaPlayer.LaunchListener with methods to be called on success or failure

void **playMedia** (*MediaInfo* mediaInfo, boolean shouldLoop, *MediaPlayer.LaunchListener* listener)

Parameters:

- mediaInfo
- shouldLoop
- listener – (optional) MediaPlayer.LaunchListener with methods to be called on success or failure

void **closeMedia** (*LaunchSession* launchSession, *ResponseListener* <Object> listener)

Parameters:

- launchSession
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

TextInputControl **getTextInputControl** ()

CapabilityPriorityLevel **getTextInputControlCapabilityLevel** ()

ServiceSubscription <*TextInputStatusListener*> **subscribeTextInputStatus** (*TextInputStatusListener* listener)

Parameters:

- listener – (optional) TextInputStatusListener with methods to be called on success or failure

void **sendText** (String input)

Parameters:

- input

void **sendKeyCode** (*KeyCode* keyCode, *ResponseListener* <Object> listener)

Parameters:

- keyCode
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **sendEnter** ()

void **sendDelete** ()

void **unsubscribe** (*URLServiceSubscription*<?> subscription)

Parameters:

- subscription

void **sendCommand** (final *ServiceCommand*<?> mCommand)

Parameters:

- mCommand

void **getPlayState** (*PlayStateListener listener*)

Get the current state of playback

Parameters:

- listener – (optional) PlayStateListener with methods to be called on success or failure

ServiceSubscription < PlayStateListener > **subscribePlayState** (*PlayStateListener listener*)

Subscribe for playback state changes

Parameters:

- listener – receives play state notifications

Returns: ServiceSubscription<PlayStateListener>

boolean **isConnectable** ()

boolean **isConnected** ()

void **connect** ()

void **disconnect** ()

void **onLoseReachability** (*DeviceServiceReachability reachability*)

Parameters:

- reachability

DIALService **getDIALService** ()

Inherited Methods

void **connect** ()

Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceListener. If the connection attempt reveals that pairing is required, the DeviceServiceListener will also be notified in that event.

void **disconnect** ()

Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceListener.

boolean **isConnected** ()

Whether the DeviceService is currently connected

boolean **isConnectable** ()

void **cancelPairing** ()

Explicitly cancels pairing in services that require pairing. In some services, this will hide a prompt that is displaying on the device.

void **sendPairingKey** (*String pairingKey*)

Will attempt to pair with the DeviceService with the provided pairingData. The failure/success will be reported back to the DeviceServiceListener.

Parameters:

- pairingKey – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

List<String> **getCapabilities** ()

boolean **hasCapability** (String *capability*)

Test to see if the capabilities array contains a given capability. See the individual Capability classes for acceptable capability values.

It is possible to append a wildcard search term `.Any` to the end of the search term. This method will return true for capabilities that match the term up to the wildcard.

Example: `Launcher.App.Any`

Parameters:

- *capability* – Capability to test against

boolean **hasAnyCapability** (String... *capabilities*) Test to see if the capabilities array contains at least one capability in a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- *capabilities* – Set of capabilities to test against

boolean **hasCapabilities** (List<String> *capabilities*)

Test to see if the capabilities array contains a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- *capabilities* – List of capabilities to test against

ServiceDescription **getServiceDescription** ()

ServiceConfig **getServiceConfig** ()

JSONObject **toJSONObject** ()

String **getServiceName** ()

Name of the DeviceService (webOS, Chromecast, etc)

void **closeLaunchSession** (*LaunchSession* *launchSession*, *ResponseListener* <Object> *listener*)

Closes the session on the first screen device. Depending on the `sessionType`, the associated service will have different ways of handling the close functionality.

Parameters:

- *launchSession* – LaunchSession to close
- *listener* – (optional) listener to be called on success/failure

Launcher **getLauncher** ()

CapabilityPriorityLevel **getLauncherCapabilityLevel** ()

void **launchAppWithInfo** (*AppInfo* *appInfo*, *AppLaunchListener* *listener*) Launch an application on the device.

Related capabilities:

- `Launcher.App`
- `Launcher.App.Params` – if launching with params

Parameters:

- appInfo – AppInfo object for the application
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void **launchApp** (String *appId*, *AppLaunchListener* listener)

Launch an application on the device.

Related capabilities:

- Launcher.App

Parameters:

- appId – ID of the application
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void **closeApp** (*LaunchSession* launchSession, *ResponseListener* <Object> listener)

Close an application on the device.

Related capabilities:

- Launcher.App.Close

Parameters:

- launchSession – LaunchSession of the target app
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **getAppList** (*AppListListener* listener)

Gets a list of all apps installed on the device.

Related capabilities:

- Launcher.App.List

Parameters:

- listener – (optional) AppListListener with methods to be called on success or failure

void **getRunningApp** (*AppInfoListener* listener)

Gets an AppInfo object for the current running app on the device.

Related capabilities:

- Launcher.RunningApp

Parameters:

- listener – (optional) AppInfoListener with methods to be called on success or failure

ServiceSubscription <*AppInfoListener*> **subscribeRunningApp** (*AppInfoListener* listener)

Subscribes to changes of the current running app. Every time the running app changes, the success block will be called with an AppInfo object for the current running app.

Related capabilities:

- Launcher.RunningApp.Subscribe

Parameters:

- listener – (optional) AppInfoListener with methods to be called on success or failure

void **getAppState** (*LaunchSession* launchSession, *AppStateListener* listener)

Gets the target app's running status and on-screen visibility.

Related capabilities:

- `Launcher.AppState`

Parameters:

- `launchSession` – `LaunchSession` of the target app
- `listener` – (optional) `AppStateListener` with methods to be called on success or failure

ServiceSubscription <*AppStateListener*> **subscribeAppState** (*LaunchSession* launchSession, *AppStateListener* listener)

Subscribes to changes of the state of the target app. Every time the app's state changes, the success block will be called with info on the app's running status and on-screen visibility.

Related capabilities:

- `Launcher.AppState.Subscribe`

Parameters:

- `launchSession` – `LaunchSession` of the target app
- `listener` – (optional) `AppStateListener` with methods to be called on success or failure

void **launchBrowser** (String url, *AppLaunchListener* listener)

Launch the web browser. Will launch deep-linked to provided URL, if supported on the target platform.

Related capabilities:

- `Launcher.Browser`
- `Launcher.Browser.Params` – if launching with url

Parameters:

- url
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void **launchYouTube** (String contentId, *AppLaunchListener* listener)

Launch YouTube app. Will launch deep-linked to provided contentId, if supported on the target platform.

Related capabilities:

- `Launcher.YouTube`
- `Launcher.YouTube.Params` – if launching with contentId

Parameters:

- `contentId` – Video id to open
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void **launchNetflix** (String contentId, *AppLaunchListener* listener)

Launch Netflix app. Will launch deep-linked to provided contentId, if supported on the target platform.

Related capabilities:

- `Launcher.Netflix`
- `Launcher.Netflix.Params` – if launching with contentId

Parameters:

- `contentId` – Video id to open
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void **launchHulu** (String *contentId*, *AppLaunchListener listener*)

Launch Hulu app. Will launch deep-linked to provided `contentId`, if supported on the target platform.

Related capabilities:

- `Launcher.Hulu`
- `Launcher.Hulu.Params` – if launching with `contentId`

Parameters:

- `contentId` – Video id to open
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void **launchAppStore** (String *appId*, *AppLaunchListener listener*)

Launch the device's app store app, optionally deep-linked to a specific app's page.

Related capabilities:

- `Launcher.AppStore`
- `Launcher.AppStore.Params`

Parameters:

- `appId` – (optional) ID of the application to show in the app store
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

MediaPlayer **getMediaPlayer** ()

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel** ()

void **getMediaInfo** (*MediaInfoListener listener*)

Parameters:

- `listener` – (optional) `MediaInfoListener` with methods to be called on success or failure

ServiceSubscription <*MediaInfoListener*> **subscribeMediaInfo** (*MediaInfoListener listener*)

Parameters:

- `listener` – (optional) `MediaInfoListener` with methods to be called on success or failure

void **displayImage** (*MediaInfo mediaInfo*, *LaunchListener listener*)

Display an image on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- `MediaPlayer.Display.Image`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- `mediaInfo` – Object of `MediaInfo` class which includes all the information about an image to display.
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void **playMedia** (*MediaInfo* `mediaInfo`, boolean *shouldLoop*, *LaunchListener* `listener`)

Play an audio or video file on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- `MediaPlayer.Play.Video`
- `MediaPlayer.Play.Audio`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- `mediaInfo` – Object of `MediaInfo` class which includes all the information about an image to display.
- `shouldLoop` – Whether to automatically loop playback
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void **closeMedia** (*LaunchSession* `launchSession`, *ResponseListener* `<Object> listener`)

Close a running media session. Because media is handled differently on different platforms, it is required to keep track of `LaunchSession` and `MediaControl` objects to control that media session in the future. `LaunchSession` will be required to close the media and `mediaControl` will be required to control the media.

Related capabilities:

- `MediaPlayer.Close`

Parameters:

- `launchSession` – `LaunchSession` object for use in closing media instance
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

MediaControl **getMediaControl** ()

Get `MediaControl` implementation

Returns: `MediaControl`

CapabilityPriorityLevel **getMediaControlCapabilityLevel** ()

Get a capability priority for current implementation

Returns: `CapabilityPriorityLevel`

void **play** (*ResponseListener* `<Object> listener`)

Send play command.

Related capabilities:

- `MediaControl.Play`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **pause** (*ResponseListener* <Object> listener)

Send pause command.

Related capabilities:

- `MediaControl.Pause`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **stop** (*ResponseListener* <Object> listener)

Send play command.

Related capabilities:

- `MediaControl.Stop`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **rewind** (*ResponseListener* <Object> listener)

Send rewind command.

Related capabilities:

- `MediaControl.Rewind`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **fastForward** (*ResponseListener* <Object> listener)

Send play command.

Related capabilities:

- `MediaControl.FastForward`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **previous** (*ResponseListener* <Object> listener)

This method is deprecated. Use `PlaylistControl::previous (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **next** (*ResponseListener* <Object> listener)

This method is deprecated. Use `PlaylistControl::next (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **seek** (long position, *ResponseListener* <Object> listener)

Seeks to a new position within the current media item

Related capabilities:

- `MediaControl.Seek`

Parameters:

- `position` – The new position, in milliseconds from the beginning of the stream
- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **getDuration** (*DurationListener listener*)

Get the current media duration in milliseconds

Parameters:

- `listener` – (optional) `DurationListener` with methods to be called on success or failure

void **getPosition** (*PositionListener listener*)

Get the current playback position in milliseconds

Parameters:

- `listener` – (optional) `PositionListener` with methods to be called on success or failure

void **getPlayState** (*PlayStateListener listener*)

Get the current state of playback

Parameters:

- `listener` – (optional) `PlayStateListener` with methods to be called on success or failure

ServiceSubscription `<PlayStateListener>` **subscribePlayState** (*PlayStateListener listener*)

Subscribe for playback state changes

Parameters:

- `listener` – receives play state notifications

Returns: `ServiceSubscription<PlayStateListener>`

KeyControl **getKeyControl** ()

CapabilityPriorityLevel **getKeyControlCapabilityLevel** ()

void **up** (*ResponseListener* `<Object>` *listener*)

Sends the up button key code to the TV.

Related capabilities:

- `KeyControl.Up`

Parameters:

- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **down** (*ResponseListener* `<Object>` *listener*)

Sends the down button key code to the TV.

Related capabilities:

- `KeyControl.Down`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **left** (*ResponseListener* <Object> *listener*)

Sends the left button key code to the TV.

Related capabilities:

- `KeyControl.Left`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **right** (*ResponseListener* <Object> *listener*)

Sends the right button key code to the TV.

Related capabilities:

- `KeyControl.Right`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **ok** (*ResponseListener* <Object> *listener*)

Sends the OK button key code to the TV.

Related capabilities:

- `KeyControl.OK`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **back** (*ResponseListener* <Object> *listener*)

Sends the back button key code to the TV.

Related capabilities:

- `KeyControl.Back`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **home** (*ResponseListener* <Object> *listener*)

Sends the home button key code to the TV.

Related capabilities:

- `KeyControl.Home`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **sendKeyCode** (*KeyCode* *keycode*, *ResponseListener* <Object> *listener*)

Sends a key code value to the TV.

Related capabilities:

- `KeyControl.Send.KeyCode`

Parameters:

- keycode
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

TextInputControl **getTextInputControl** ()

CapabilityPriorityLevel **getTextInputControlCapabilityLevel** ()

ServiceSubscription *<TextInputStatusListener>* **subscribeTextInputStatus** (*TextInputStatusListener* listener)

Subscribe to information about the current text field.

Related capabilities:

- `TextInputControl.Subscribe`

Parameters:

- listener – (optional) `TextInputStatusListener` with methods to be called on success or failure

void **sendText** (String *input*)

Send text to the current text field.

Related capabilities:

- `TextInputControl.Send.Text`

Parameters:

- *input*

void **sendEnter** ()

Send enter key to the current text field.

Related capabilities:

- `TextInputControl.Send.Enter`

void **sendDelete** ()

Send delete event to the current text field.

Related capabilities:

- `TextInputControl.Send.Delete`

void **onLoseReachability** (*DeviceServiceReachability* *reachability*)

Parameters:

- *reachability*

void **unsubscribe** (*URLServiceSubscription<?>* *subscription*)

Parameters:

- *subscription*

void **sendCommand** (*ServiceCommand<?>* *command*)

Parameters:

- *command*

WebOSTVService

`com.connectsdk.service.WebOSTVService`

extends DeviceService

WebOSTVService provides capabilities for LG Smart TVs running webOS (model year 2014). The second screen gateway running on the webOS provides different capabilities based on whether pairing is enabled or not.

- Web app launching & two-way communication
- App launching
- Media playback
- Media control
- Volume control
- Text input control*
- Key control (fiveway)*
- Mouse control*
- Power control*
- TV control (change channels, get channel info)*
- External input control*
- Toast control*

* = requires pairing

Commands & subscriptions on webOS occur over a WebSocket connection.

webOS Version History

The following version numbers represent the version of webOS released for LG Smart TVs. The version numbers are associated with any changes to the platform's second screen APIs in that particular version.

4.0.0

- Initial release

4.0.1

- No changes

4.0.2

- Added app-to-app support
- Added the ability to request pin or prompt pairing

4.0.3

- Fixed a subscription bug in app-to-app

Properties

final String ID = “webOS TV”

final String[] kWebOSTVServiceOpenPermissions = { “LAUNCH”, “LAUNCH_WEBAPP”, “APP_TO_APP”, “CONTROL_AUDIO”, “CONTROL_INPUT_MEDIA_PLAYBACK” }

final String[] kWebOSTVServiceProtectedPermissions = { “CONTROL_POWER”, “READ_INSTALLED_APPS”, “CONTROL_DISPLAY”, “CONTROL_INPUT_JOYSTICK”, “CONTROL_INPUT_MEDIA_RECORDING”, “CONTROL_INPUT_TV”, “READ_INPUT_DEVICE_LIST”, “READ_NETWORK_STATE”, “READ_TV_CHANNEL_LIST”, “WRITE_NOTIFICATION_TOAST” }

final String[] kWebOSTVServicePersonalActivityPermissions = { “CONTROL_INPUT_TEXT”, “CONTROL_MOUSE_AND_KEYBOARD”, “READ_CURRENT_CHANNEL”, “READ_RUNNING_APPS” }

Inner Classes

- ACRAuthTokenListener
- LaunchPointsListener
- SecureAccessTestListener
- ServiceInfoListener
- SystemInfoListener
- WebOSTVServicePermission

Methods

WebOSTVService (ServiceDescription *serviceDescription*, ServiceConfig *serviceConfig*)

Parameters:

- *serviceDescription*
- *serviceConfig*

void **setPairingType** (*PairingType* *pairingType*)

Parameters:

- *pairingType*

CapabilityPriorityLevel **getPriorityLevel** (Class<?extends CapabilityMethods > *clazz*)

Parameters:

- *clazz*

void **setServiceDescription** (ServiceDescription *serviceDescription*)

Parameters:

- *serviceDescription*

boolean **isConnected** ()

void **connect** ()

void **disconnect** ()

void **cancelPairing** ()

Launcher **getLauncher** ()

CapabilityPriorityLevel **getLauncherCapabilityLevel** ()

void **launchApp** (String *appId*, *AppLaunchListener* *listener*)

Parameters:

- *appId*
- *listener* – (optional) *AppLaunchListener* with methods to be called on success or failure

void **launchAppWithInfo** (*AppInfo* *appInfo*, *Launcher.AppLaunchListener* *listener*)

Parameters:

- *appInfo*
- *listener* – (optional) *Launcher.AppLaunchListener* with methods to be called on success or failure

void **launchAppWithInfo** (final *AppInfo* *appInfo*, Object *params*, final *Launcher.AppLaunchListener* *listener*)

Parameters:

- *appInfo*
- *params*
- *listener* – (optional) final *Launcher.AppLaunchListener* with methods to be called on success or failure

void **launchBrowser** (String *url*, final *Launcher.AppLaunchListener* *listener*)

Parameters:

- *url*
- *listener* – (optional) final *Launcher.AppLaunchListener* with methods to be called on success or failure

void **launchYouTube** (String *contentId*, *Launcher.AppLaunchListener* *listener*)

Parameters:

- *contentId*
- *listener* – (optional) *Launcher.AppLaunchListener* with methods to be called on success or failure

void **launchYouTube** (final String *contentId*, float *startTime*, final *AppLaunchListener* *listener*)

Parameters:

- *contentId*
- *startTime*
- *listener* – (optional) final *AppLaunchListener* with methods to be called on success or failure

void **launchHulu** (String *contentId*, *Launcher.AppLaunchListener* *listener*)

Parameters:

- *contentId*
- *listener* – (optional) *Launcher.AppLaunchListener* with methods to be called on success or failure

void **launchNetflix** (String *contentId*, *Launcher.AppLaunchListener* *listener*)

Parameters:

- `contentId`
- `listener` – (optional) `Launcher.AppLaunchListener` with methods to be called on success or failure

void **launchAppStore** (String *appId*, *AppLaunchListener listener*)

Parameters:

- `appId`
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void **closeApp** (*LaunchSession launchSession*, *ResponseListener <Object> listener*)

Parameters:

- `launchSession`
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **getAppList** (final *AppListListener listener*)

Parameters:

- `listener` – (optional) final `AppListListener` with methods to be called on success or failure

void **getRunningApp** (*AppInfoListener listener*)

Parameters:

- `listener` – (optional) `AppInfoListener` with methods to be called on success or failure

ServiceSubscription <AppInfoListener> **subscribeRunningApp** (*AppInfoListener listener*)

Parameters:

- `listener` – (optional) `AppInfoListener` with methods to be called on success or failure

void **getAppState** (*LaunchSession launchSession*, *AppStateListener listener*)

Parameters:

- `launchSession`
- `listener` – (optional) `AppStateListener` with methods to be called on success or failure

ServiceSubscription <AppStateListener> **subscribeAppState** (*LaunchSession launchSession*, *AppStateListener listener*)

Parameters:

- `launchSession`
- `listener` – (optional) `AppStateListener` with methods to be called on success or failure

ToastControl **getToastControl** ()

CapabilityPriorityLevel **getToastControlCapabilityLevel** ()

void **showToast** (String *message*, *ResponseListener <Object> listener*)

Parameters:

- `message`
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **showToast** (String *message*, String *iconData*, String *iconExtension*, *ResponseListener <Object> listener*)

Parameters:

- message
- iconData
- iconExtension
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **showClickableToastForApp** (String *message*, *AppInfo* *appInfo*, `JSONObject` *params*, *ResponseListener* <Object> *listener*)

Parameters:

- message
- appInfo
- params
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **showClickableToastForApp** (String *message*, *AppInfo* *appInfo*, `JSONObject` *params*, String *iconData*, String *iconExtension*, *ResponseListener* <Object> *listener*)

Parameters:

- message
- appInfo
- params
- iconData
- iconExtension
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **showClickableToastForURL** (String *message*, String *url*, *ResponseListener* <*and-responselister*> <Object> *listener*)

Parameters:

- message
- url
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **showClickableToastForURL** (String *message*, String *url*, String *iconData*, String *iconExtension*, *ResponseListener* <*and-responselister*> <Object> *listener*)

Parameters:

- message
- url
- iconData
- iconExtension
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

VolumeControl **getVolumeControl** ()

CapabilityPriorityLevel **getVolumeControlCapabilityLevel** ()

void **volumeUp** ()

void **volumeUp** (*ResponseListener* <Object> *listener*)

Parameters:

- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void **volumeDown** ()

void **volumeDown** (*ResponseListener* <Object> *listener*)

Parameters:

- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void **setVolume** (int *volume*)

Parameters:

- *volume*

void **setVolume** (float *volume*, *ResponseListener* <Object> *listener*)

Parameters:

- *volume*
- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void **getVolume** (*VolumeListener* *listener*)

Parameters:

- *listener* – (optional) *VolumeListener* with methods to be called on success or failure

ServiceSubscription <*VolumeListener*> **subscribeVolume** (*VolumeListener* *listener*)

Parameters:

- *listener* – (optional) *VolumeListener* with methods to be called on success or failure

void **setMute** (boolean *isMute*, *ResponseListener* <Object> *listener*)

Parameters:

- *isMute*
- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void **getMute** (*MuteListener* *listener*)

Parameters:

- *listener* – (optional) *MuteListener* with methods to be called on success or failure

ServiceSubscription <*MuteListener*> **subscribeMute** (*MuteListener* *listener*)

Parameters:

- *listener* – (optional) *MuteListener* with methods to be called on success or failure

void **getVolumeStatus** (*VolumeStatusListener* *listener*)

Parameters:

- *listener* – (optional) *VolumeStatusListener* with methods to be called on success or failure

ServiceSubscription <*VolumeStatusListener*> **subscribeVolumeStatus** (*VolumeStatusListener* *listener*)

Parameters:

- listener – (optional) VolumeStatusListener with methods to be called on success or failure

MediaPlayer **getMediaPlayer** ()

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel** ()

void **getMediaInfo** (*MediaInfoListener* listener)

Parameters:

- listener – (optional) MediaInfoListener with methods to be called on success or failure

ServiceSubscription <*MediaInfoListener*> **subscribeMediaInfo** (*MediaInfoListener* listener)

Parameters:

- listener – (optional) MediaInfoListener with methods to be called on success or failure

void **displayImage** (final String *url*, final String *mimeType*, final String *title*, final String *description*, final String *iconSrc*, final *MediaPlayer.LaunchListener* listener)

Parameters:

- url
- mimeType
- title
- description
- iconSrc
- listener – (optional) final MediaPlayer.LaunchListener with methods to be called on success or failure

void **displayImage** (*MediaInfo* mediaInfo, *MediaPlayer.LaunchListener* listener)

Parameters:

- mediaInfo
- listener – (optional) MediaPlayer.LaunchListener with methods to be called on success or failure

void **playMedia** (String *url*, String *mimeType*, String *title*, String *description*, String *iconSrc*, boolean *shouldLoop*, *MediaPlayer.LaunchListener* listener)

Parameters:

- url
- mimeType
- title
- description
- iconSrc
- shouldLoop
- listener – (optional) MediaPlayer.LaunchListener with methods to be called on success or failure

void **playMedia** (*MediaInfo* mediaInfo, boolean *shouldLoop*, *MediaPlayer.LaunchListener* listener)

Parameters:

- mediaInfo

- shouldLoop
- listener – (optional) MediaPlayer.LaunchListener with methods to be called on success or failure

void **closeMedia** (*LaunchSession* launchSession, *ResponseListener* <Object> listener)

Parameters:

- launchSession
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

MediaControl **getMediaControl** ()

Get MediaControl implementation

Returns: MediaControl

CapabilityPriorityLevel **getMediaControlCapabilityLevel** ()

Get a capability priority for current implementation

Returns: CapabilityPriorityLevel

void **play** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **pause** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **stop** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **rewind** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **fastForward** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **previous** (*ResponseListener* <Object> listener)

This method is deprecated. Use `PlaylistControl::previous (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **next** (*ResponseListener* <Object> listener)

This method is deprecated. Use `PlaylistControl::next (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **seek** (long *position*, *ResponseListener* <Object> *listener*)

Parameters:

- *position* – The new position, in milliseconds from the beginning of the stream
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **getDuration** (*DurationListener* *listener*)

Get the current media duration in milliseconds

Parameters:

- listener – (optional) `DurationListener` with methods to be called on success or failure

void **getPosition** (*PositionListener* *listener*)

Get the current playback position in milliseconds

Parameters:

- listener – (optional) `PositionListener` with methods to be called on success or failure

TVControl **getTVControl** ()

CapabilityPriorityLevel **getTVControlCapabilityLevel** ()

void **channelUp** ()

void **channelUp** (*ResponseListener* <Object> *listener*)

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **channelDown** ()

void **channelDown** (*ResponseListener* <Object> *listener*)

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **setChannel** (*ChannelInfo* *channelInfo*, *ResponseListener* <Object> *listener*)

Sets current channel

Parameters:

- *channelInfo* – must not be null
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **setChannelById** (String *channelId*)

Parameters:

- *channelId*

void **setChannelById** (String *channelId*, *ResponseListener* <and-responselister> <Object> *listener*)

Parameters:

- *channelId*
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **getCurrentChannel** (*ChannelListener* *listener*)

Parameters:

- listener – (optional) ChannelListener with methods to be called on success or failure

ServiceSubscription <ChannelListener> **subscribeCurrentChannel** (*ChannelListener* listener)

Parameters:

- listener – (optional) ChannelListener with methods to be called on success or failure

void **getChannelList** (*ChannelListListener* listener)

Parameters:

- listener – (optional) ChannelListListener with methods to be called on success or failure

ServiceSubscription <ChannelListListener> **subscribeChannelList** (final *ChannelListListener* listener)

Parameters:

- listener – (optional) final ChannelListListener with methods to be called on success or failure

void **getChannelCurrentProgramInfo** (*ProgramInfoListener* listener)

Parameters:

- listener – (optional) ProgramInfoListener with methods to be called on success or failure

ServiceSubscription <ProgramInfoListener> **subscribeChannelCurrentProgramInfo** (*ProgramInfoListener* listener)

Parameters:

- listener – (optional) ProgramInfoListener with methods to be called on success or failure

void **getProgramInfo** (*ProgramInfoListener* listener)

Parameters:

- listener – (optional) ProgramInfoListener with methods to be called on success or failure

ServiceSubscription <ProgramInfoListener> **subscribeProgramInfo** (*ProgramInfoListener* listener)

Parameters:

- listener – (optional) ProgramInfoListener with methods to be called on success or failure

void **getProgramList** (*ProgramListListener* listener)

Parameters:

- listener – (optional) ProgramListListener with methods to be called on success or failure

ServiceSubscription <ProgramListListener> **subscribeProgramList** (*ProgramListListener* listener)

Parameters:

- listener – (optional) ProgramListListener with methods to be called on success or failure

void **set3DEnabled** (final boolean *enabled*, final *ResponseListener* <Object> listener)

Parameters:

- enabled
- listener – (optional) final ResponseListener< Object > with methods to be called on success or failure

void **get3DEnabled** (final *State3DModeListener* listener)

Parameters:

- listener – (optional) final *State3DModeListener* with methods to be called on success or failure

ServiceSubscription <*State3DModeListener*> **subscribe3DEnabled** (final *State3DModeListener* listener)

Parameters:

- listener – (optional) final *State3DModeListener* with methods to be called on success or failure

ExternalInputControl **getExternalInput** ()

CapabilityPriorityLevel **getExternalInputControlPriorityLevel** ()

void **launchInputPicker** (final *AppLaunchListener* listener)

Parameters:

- listener – (optional) final *AppLaunchListener* with methods to be called on success or failure

void **closeInputPicker** (*LaunchSession* launchSession, *ResponseListener* <Object> listener)

Parameters:

- launchSession
- listener – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void **getExternalInputList** (final *ExternalInputListListener* listener)

Parameters:

- listener – (optional) final *ExternalInputListListener* with methods to be called on success or failure

void **setExternalInput** (*ExternalInputInfo* externalInputInfo, final *ResponseListener* <Object> listener)

Parameters:

- externalInputInfo
- listener – (optional) final *ResponseListener*< Object > with methods to be called on success or failure

MouseControl **getMouseControl** ()

CapabilityPriorityLevel **getMouseControlCapabilityLevel** ()

void **connectMouse** ()

void **disconnectMouse** ()

void **click** ()

void **move** (final double dx, final double dy)

Parameters:

- dx
- dy

void **move** (*PointF* diff)

Parameters:

- diff

void **scroll** (final double dx, final double dy)

Parameters:

- dx

- dy

void **scroll** (PointF *diff*)

Parameters:

- diff

TextInputControl **getTextInputControl** ()

CapabilityPriorityLevel **getTextInputControlCapabilityLevel** ()

ServiceSubscription <*TextInputStatusListener*> **subscribeTextInputStatus** (*TextInputStatusListener* *listener*)

Parameters:

- listener – (optional) *TextInputStatusListener* with methods to be called on success or failure

void **sendText** (String *input*)

Parameters:

- input

void **sendKeyCode** (*KeyCode* *keycode*, *ResponseListener* <Object> *listener*)

Parameters:

- keycode
- listener – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void **sendEnter** ()

void **sendDelete** ()

PowerControl **getPowerControl** ()

CapabilityPriorityLevel **getPowerControlCapabilityLevel** ()

void **powerOff** (*ResponseListener* <Object> *listener*)

Parameters:

- listener – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void **powerOn** (*ResponseListener* <Object> *listener*)

Parameters:

- listener – (optional) *ResponseListener*< Object > with methods to be called on success or failure

KeyControl **getKeyControl** ()

CapabilityPriorityLevel **getKeyControlCapabilityLevel** ()

void **up** (*ResponseListener* <Object> *listener*)

Parameters:

- listener – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void **down** (*ResponseListener* <Object> *listener*)

Parameters:

- listener – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void **left** (*ResponseListener* <Object> *listener*)

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **right** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **ok** (final *ResponseListener* <Object> listener)

Parameters:

- listener – (optional) final `ResponseListener<Object>` with methods to be called on success or failure

void **back** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **home** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

WebAppLauncher **getWebAppLauncher** ()

CapabilityPriorityLevel **getWebAppLauncherCapabilityLevel** ()

void **launchWebApp** (final String *webAppId*, final *WebAppSession.LaunchListener* listener)

Parameters:

- *webAppId*
- listener – (optional) final `WebAppSession.LaunchListener` with methods to be called on success or failure

void **launchWebApp** (String *webAppId*, boolean *relaunchIfRunning*, *WebAppSession.LaunchListener* listener)

Parameters:

- *webAppId*
- *relaunchIfRunning*
- listener – (optional) `WebAppSession.LaunchListener` with methods to be called on success or failure

void **launchWebApp** (final String *webAppId*, final `JSONObject` *params*, final *WebAppSession.LaunchListener* listener)

Parameters:

- *webAppId*
- *params*
- listener – (optional) final `WebAppSession.LaunchListener` with methods to be called on success or failure

void **launchWebApp** (final String *webAppId*, final `JSONObject` *params*, boolean *relaunchIfRunning*, final *WebAppSession.LaunchListener* listener)

Parameters:

- *webAppId*

- params
- relaunchIfRunning
- listener – (optional) final `WebAppSession.LaunchListener` with methods to be called on success or failure

void **closeWebApp** (*LaunchSession* launchSession, final *ResponseListener* <Object> listener)

Parameters:

- launchSession
- listener – (optional) final `ResponseListener< Object >` with methods to be called on success or failure

void **connectToWebApp** (final `WebOSWebAppSession` webAppSession, final boolean joinOnly, final *ResponseListener* <Object> connectionListener)

Parameters:

- webAppSession
- joinOnly
- connectionListener

void **pinWebApp** (String webAppId, final *ResponseListener* <Object> listener)

Parameters:

- webAppId
- listener – (optional) final `ResponseListener< Object >` with methods to be called on success or failure

void **unPinWebApp** (String webAppId, final *ResponseListener* <Object> listener)

Parameters:

- webAppId
- listener – (optional) final `ResponseListener< Object >` with methods to be called on success or failure

void **isWebAppPinned** (String webAppId, *WebAppPinStatusListener* listener)

Parameters:

- webAppId
- listener – (optional) `WebAppPinStatusListener` with methods to be called on success or failure

ServiceSubscription <*WebAppPinStatusListener*> **subscribeIsWebAppPinned** (String webAppId, *WebAppPinStatusListener* listener)

Parameters:

- webAppId
- listener – (optional) `WebAppPinStatusListener` with methods to be called on success or failure

void **joinApp** (String appId, *WebAppSession.LaunchListener* listener)

Parameters:

- appId

- listener – (optional) `WebAppSession.LaunchListener` with methods to be called on success or failure

void **connectToApp** (String *appId*, final *WebAppSession.LaunchListener* *listener*)

Parameters:

- *appId*
- listener – (optional) final `WebAppSession.LaunchListener` with methods to be called on success or failure

void **joinWebApp** (final *LaunchSession* *webAppLaunchSession*, final *WebAppSession.LaunchListener* *listener*)

Parameters:

- *webAppLaunchSession*
- listener – (optional) final `WebAppSession.LaunchListener` with methods to be called on success or failure

void **joinWebApp** (String *webAppId*, *WebAppSession.LaunchListener* *listener*)

Parameters:

- *webAppId*
- listener – (optional) `WebAppSession.LaunchListener` with methods to be called on success or failure

void **sendMessage** (String *message*, *LaunchSession* *launchSession*, *ResponseListener* <Object> *listener*)

Parameters:

- *message*
- *launchSession*
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **sendMessage** (JSONObject *message*, *LaunchSession* *launchSession*, *ResponseListener* <Object> *listener*)

Parameters:

- *message*
- *launchSession*
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **getServiceInfo** (final *ServiceInfoListener* *listener*)

Parameters:

- listener – (optional) final `ServiceInfoListener` with methods to be called on success or failure

void **getSystemInfo** (final *SystemInfoListener* *listener*)

Parameters:

- listener – (optional) final `SystemInfoListener` with methods to be called on success or failure

void **secureAccessTest** (final *SecureAccessTestListener* *listener*)

Parameters:

- listener – (optional) final `SecureAccessTestListener` with methods to be called on success or failure

void **getACRAuthToken** (final *ACRAuthTokenListener* *listener*)

Parameters:

- listener – (optional) final ACRAuthTokenListener with methods to be called on success or failure

void **getLaunchPoints** (final LaunchPointsListener *listener*)

Parameters:

- listener – (optional) final LaunchPointsListener with methods to be called on success or failure

PlaylistControl **getPlaylistControl** ()

CapabilityPriorityLevel **getPlaylistControlCapabilityLevel** ()

void **jumpToTrack** (long *index*, *ResponseListener* <Object> *listener*)

Play a track specified by index in the playlist

Parameters:

- index – index in the playlist, it starts from zero like index of array
- listener – optional response listener

void **setPlayMode** (*PlayMode* *playMode*, *ResponseListener* <Object> *listener*)

Set order of playing tracks

Parameters:

- playMode
- listener – optional response listener

void **sendCommand** (*ServiceCommand*<?> *command*)

Parameters:

- command

void **unsubscribe** (*URLServiceSubscription*<?> *subscription*)

Parameters:

- subscription

List<String> **getPermissions** ()

void **setPermissions** (List<String> *permissions*)

Parameters:

- permissions

void **getPlayState** (*PlayStateListener* *listener*)

Get the current state of playback

Parameters:

- listener – (optional) PlayStateListener with methods to be called on success or failure

ServiceSubscription <*PlayStateListener*> **subscribePlayState** (*PlayStateListener* *listener*)

Subscribe for playback state changes

Parameters:

- listener – receives play state notifications

Returns: ServiceSubscription<PlayStateListener>

boolean **isConnectable** ()

void **sendPairingKey** (String *pairingKey*)

Parameters:

- *pairingKey*

static DiscoveryFilter **discoveryFilter** ()

Inherited Methods

void connect () Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceListener. If the connection attempt reveals that pairing is required, the DeviceServiceListener will also be notified in that event.

void disconnect () Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceListener.

boolean isConnected () Whether the DeviceService is currently connected

boolean **isConnectable** ()

void cancelPairing () Explicitly cancels pairing in services that require pairing. In some services, this will hide a prompt that is displaying on the device.

void sendPairingKey (String *pairingKey*) Will attempt to pair with the DeviceService with the provided pairing-Data. The failure/success will be reported back to the DeviceServiceListener.

Parameters:

- *pairingKey* – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

List<String> **getCapabilities** ()

boolean hasCapability (String *capability*) Test to see if the capabilities array contains a given capability. See the individual Capability classes for acceptable capability values.

It is possible to append a wildcard search term `.Any` to the end of the search term. This method will return true for capabilities that match the term up to the wildcard.

Example: `Launcher.App.Any`

Parameters:

- *capability* – Capability to test against

boolean hasAnyCapability (String... *capabilities*) Test to see if the capabilities array contains at least one capability in a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- *capabilities* – Set of capabilities to test against

boolean hasCapabilities (List<String> *capabilities*) Test to see if the capabilities array contains a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability`: for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – List of capabilities to test against

ServiceDescription **getServiceDescription** ()

ServiceConfig **getServiceConfig** ()

JSONObject **toJSONObject** ()

String **getServiceName** ()

Name of the DeviceService (webOS, Chromecast, etc)

void **closeLaunchSession** (*LaunchSession launchSession*, *ResponseListener <Object> listener*)

Closes the session on the first screen device. Depending on the sessionType, the associated service will have different ways of handling the close functionality.

Parameters:

- launchSession – LaunchSession to close
- listener – (optional) listener to be called on success/failure

Launcher **getLauncher** ()

CapabilityPriorityLevel **getLauncherCapabilityLevel** ()

void **launchAppWithInfo** (*AppInfo appInfo*, *AppLaunchListener listener*)

Launch an application on the device.

Related capabilities:

- Launcher.App
- Launcher.App.Params – if launching with params

Parameters:

- appInfo – AppInfo object for the application
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void **launchApp** (String *appId*, *AppLaunchListener listener*)

Launch an application on the device.

Related capabilities:

- Launcher.App

Parameters:

- appId – ID of the application
- listener – (optional) AppLaunchListener with methods to be called on success or failure

void **closeApp** (*LaunchSession launchSession*, *ResponseListener <Object> listener*)

Close an application on the device.

Related capabilities:

- Launcher.App.Close

Parameters:

- launchSession – LaunchSession of the target app
- listener – (optional) ResponseListener<Object> with methods to be called on success or failure

void **getAppList** (*AppListListener* listener)

Gets a list of all apps installed on the device.

Related capabilities:

- `Launcher.App.List`

Parameters:

- listener – (optional) `AppListListener` with methods to be called on success or failure

void **getRunningApp** (*AppInfoListener* listener)

Gets an `AppInfo` object for the current running app on the device.

Related capabilities:

- `Launcher.RunningApp`

Parameters:

- listener – (optional) `AppInfoListener` with methods to be called on success or failure

ServiceSubscription <*AppInfoListener*> **subscribeRunningApp** (*AppInfoListener* listener)

Subscribes to changes of the current running app. Every time the running app changes, the success block will be called with an `AppInfo` object for the current running app.

Related capabilities:

- `Launcher.RunningApp.Subscribe`

Parameters:

- listener – (optional) `AppInfoListener` with methods to be called on success or failure

void **getAppState** (*LaunchSession* launchSession, *AppStateListener* listener)

Gets the target app's running status and on-screen visibility.

Related capabilities:

- `Launcher.AppState`

Parameters:

- launchSession – `LaunchSession` of the target app
- listener – (optional) `AppStateListener` with methods to be called on success or failure

ServiceSubscription <*AppStateListener*> **subscribeAppState** (*LaunchSession* launchSession, *AppStateListener* listener)

Subscribes to changes of the state of the target app. Every time the app's state changes, the success block will be called with info on the app's running status and on-screen visibility.

Related capabilities:

- `Launcher.AppState.Subscribe`

Parameters:

- launchSession – `LaunchSession` of the target app
- listener – (optional) `AppStateListener` with methods to be called on success or failure

void **launchBrowser** (*String* url, *AppLaunchListener* listener)

Launch the web browser. Will launch deep-linked to provided URL, if supported on the target platform.

Related capabilities:

- `Launcher.Browser`
- `Launcher.Browser.Params` – if launching with url

Parameters:

- url
- listener – (optional) `AppLaunchListener` with methods to be called on success or failure

void **launchYouTube** (String *contentId*, *AppLaunchListener* listener)

Launch YouTube app. Will launch deep-linked to provided `contentId`, if supported on the target platform.

Related capabilities:

- `Launcher.YouTube`
- `Launcher.YouTube.Params` – if launching with `contentId`

Parameters:

- `contentId` – Video id to open
- listener – (optional) `AppLaunchListener` with methods to be called on success or failure

void **launchNetflix** (String *contentId*, *AppLaunchListener* listener)

Launch Netflix app. Will launch deep-linked to provided `contentId`, if supported on the target platform.

Related capabilities:

- `Launcher.Netflix`
- `Launcher.Netflix.Params` – if launching with `contentId`

Parameters:

- `contentId` – Video id to open
- listener – (optional) `AppLaunchListener` with methods to be called on success or failure

void **launchHulu** (String *contentId*, *AppLaunchListener* listener)

Launch Hulu app. Will launch deep-linked to provided `contentId`, if supported on the target platform.

Related capabilities:

- `Launcher.Hulu`
- `Launcher.Hulu.Params` – if launching with `contentId`

Parameters:

- `contentId` – Video id to open
- listener – (optional) `AppLaunchListener` with methods to be called on success or failure

void **launchAppStore** (String *appId*, *AppLaunchListener* listener)

Launch the device's app store app, optionally deep-linked to a specific app's page.

Related capabilities:

- `Launcher.AppStore`
- `Launcher.AppStore.Params`

Parameters:

- `appId` – (optional) ID of the application to show in the app store
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

MediaControl **getMediaControl** ()

Get `MediaControl` implementation

Returns: `MediaControl`

CapabilityPriorityLevel **getMediaControlCapabilityLevel** ()

Get a capability priority for current implementation

Returns: `CapabilityPriorityLevel`

void **play** (*ResponseListener* <Object> *listener*)

Send play command.

Related capabilities:

- `MediaControl.Play`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **pause** (*ResponseListener* <Object> *listener*)

Send pause command.

Related capabilities:

- `MediaControl.Pause`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **stop** (*ResponseListener* <Object> *listener*)

Send play command.

Related capabilities:

- `MediaControl.Stop`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **rewind** (*ResponseListener* <Object> *listener*)

Send rewind command.

Related capabilities:

- `MediaControl.Rewind`

Parameters:

- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **fastForward** (*ResponseListener* <Object> *listener*)

Send play command.

Related capabilities:

- `MediaControl.FastForward`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **previous** (*ResponseListener* <Object> listener)

This method is deprecated. Use `PlaylistControl::previous (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **next** (*ResponseListener* <Object> listener)

This method is deprecated. Use `PlaylistControl::next (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **seek** (long position, *ResponseListener* <Object> listener)

Seeks to a new position within the current media item

Related capabilities:

- `MediaControl.Seek`

Parameters:

- position – The new position, in milliseconds from the beginning of the stream
- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **getDuration** (*DurationListener* listener)

Get the current media duration in milliseconds

Parameters:

- listener – (optional) `DurationListener` with methods to be called on success or failure

void **getPosition** (*PositionListener* listener) Get the current playback position in milliseconds

Parameters:

- listener – (optional) `PositionListener` with methods to be called on success or failure

void **getPlayState** (*PlayStateListener* listener) Get the current state of playback

Parameters:

- listener – (optional) `PlayStateListener` with methods to be called on success or failure

ServiceSubscription <*PlayStateListener*> **subscribePlayState** (*PlayStateListener* listener) Subscribe for playback state changes

Parameters:

- listener – receives play state notifications

Returns: `ServiceSubscription<PlayStateListener>`

MediaPlayer **getMediaPlayer** ()

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel** ()

void **getMediaInfo** (*MediaInfoListener* listener)

Parameters:

- listener – (optional) `MediaInfoListener` with methods to be called on success or failure

ServiceSubscription <*MediaInfoListener*> **subscribeMediaInfo** (*MediaInfoListener* listener)

Parameters:

- listener – (optional) `MediaInfoListener` with methods to be called on success or failure

void **displayImage** (*MediaInfo* mediaInfo, `LaunchListener` listener)

Display an image on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- `MediaPlayer.Display.Image`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- mediaInfo – Object of `MediaInfo` class which includes all the information about an image to display.
- listener – (optional) `LaunchListener` with methods to be called on success or failure

void **playMedia** (*MediaInfo* mediaInfo, boolean *shouldLoop*, `LaunchListener` listener)

Play an audio or video file on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- `MediaPlayer.Play.Video`
- `MediaPlayer.Play.Audio`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- mediaInfo – Object of `MediaInfo` class which includes all the information about an image to display.
- shouldLoop – Whether to automatically loop playback
- listener – (optional) `LaunchListener` with methods to be called on success or failure

void **closeMedia** (*LaunchSession* launchSession, *ResponseListener* <Object> listener)

Close a running media session. Because media is handled differently on different platforms, it is required to keep track of `LaunchSession` and `MediaControl` objects to control that media session in the future. `LaunchSession` will be required to close the media and `mediaControl` will be required to control the media.

Related capabilities:

- `MediaPlayer.Close`

Parameters:

- `launchSession` – `LaunchSession` object for use in closing media instance
- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

VolumeControl **getVolumeControl** ()

CapabilityPriorityLevel **getVolumeControlCapabilityLevel** ()

void **volumeUp** (*ResponseListener* <Object> *listener*)

Sends the volume up command to the device.

Related capabilities:

- `VolumeControl.UpDown`

Parameters:

- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **volumeDown** (*ResponseListener* <Object> *listener*)

Sends the volume down command to the device.

Related capabilities:

- `VolumeControl.UpDown`

Parameters:

- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **setVolume** (float *volume*, *ResponseListener* <Object> *listener*)

Set the volume of the device.

Related capabilities:

- `VolumeControl.Set`

Parameters:

- `volume` – Volume as a float between 0.0 and 1.0
- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **getVolume** (*VolumeListener* *listener*)

Get the current volume of the device.

Related capabilities:

- `VolumeControl.Get`

Parameters:

- `listener` – (optional) `VolumeListener` with methods to be called on success or failure

void **setMute** (boolean *isMute*, *ResponseListener* <Object> *listener*)

Set the current volume.

Related capabilities:

- `VolumeControl.Mute.Set`

Parameters:

- isMute
- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **getMute** (*MuteListener* listener)

Get the current mute state.

Related capabilities:

- `VolumeControl.Mute.Get`

Parameters:

- listener – (optional) `MuteListener` with methods to be called on success or failure

ServiceSubscription <VolumeListener> **subscribeVolume** (*VolumeListener* listener)

Subscribe to the volume on the TV.

Related capabilities:

- `VolumeControl.Subscribe`

Parameters:

- listener – (optional) `VolumeListener` with methods to be called on success or failure

ServiceSubscription <MuteListener> **subscribeMute** (*MuteListener* listener)

Subscribe to the mute state on the TV.

Related capabilities:

- `VolumeControl.Mute.Subscribe`

Parameters:

- listener – (optional) `MuteListener` with methods to be called on success or failure

TVControl **getTVControl** ()

CapabilityPriorityLevel **getTVControlCapabilityLevel** ()

void **channelUp** (*ResponseListener <Object>* listener)

Sends a channel up command to the TV.

Related capabilities:

- `TVControl.Channel.Up`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **channelDown** (*ResponseListener <Object>* listener)

Sends a channel down command to the TV.

Related capabilities:

- `TVControl.Channel.Down`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **setChannel** (*ChannelInfo* channelNumber, *ResponseListener <Object>* listener)

Sets the current channel to the channel provided by the `ChannelInfo` object provided.

Related capabilities:

- `TVControl.Channel.Set`

Parameters:

- `channelNumber`
- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **getCurrentChannel** (*ChannelListener listener*)

Gets the current channel info from the TV.

Related capabilities:

- `TVControl.Channel.Get`

Parameters:

- `listener` – (optional) `ChannelListener` with methods to be called on success or failure

ServiceSubscription <ChannelListener> **subscribeCurrentChannel** (*ChannelListener listener*)

Subscribes to any changes in the current channel. Each time the channel is changed, the new channel's info will be provided to the success callback.

Related capabilities:

- `TVControl.Channel.Subscribe`

Parameters:

- `listener` – (optional) `ChannelListener` with methods to be called on success or failure

void **getChannelList** (*ChannelListListener listener*)

Get a list of available channels from the TV.

Related capabilities:

- `TVControl.Channel.List`

Parameters:

- `listener` – (optional) `ChannelListListener` with methods to be called on success or failure

void **getProgramInfo** (*ProgramInfoListener listener*)

Gets the current program info from the TV.

Related capabilities:

- `TVControl.Program.Get`

Parameters:

- `listener` – (optional) `ProgramInfoListener` with methods to be called on success or failure

ServiceSubscription <ProgramInfoListener> **subscribeProgramInfo** (*ProgramInfoListener listener*)

Subscribes to any changes in the current program. Each time the channel is changed or a new program starts, the new program's info will be provided to the success callback.

Related capabilities:

- `TVControl.Program.Subscribe`

Parameters:

- `listener` – (optional) `ProgramInfoListener` with methods to be called on success or failure

void **getProgramList** (*ProgramListListener listener*)

Gets a list of all programs scheduled to play on the current channel.

Related capabilities:

- `TVControl.Program.List`

Parameters:

- `listener` – (optional) `ProgramListListener` with methods to be called on success or failure

ServiceSubscription <ProgramListListener> **subscribeProgramList** (*ProgramListListener listener*)

Subscribes to any changes in the current program. Each time the channel is changed or a new program starts, the new program's info will be provided to the success callback.

Related capabilities:

- `TVControl.Program.List.Subscribe`

Parameters:

- `listener` – (optional) `ProgramListListener` with methods to be called on success or failure

void **get3DEnabled** (*State3DModeListener listener*)

Gets the current 3D status of the TV.

Related capabilities:

- `TVControl.3D.Get`

Parameters:

- `listener` – (optional) `State3DModeListener` with methods to be called on success or failure

void **set3DEnabled** (boolean *enabled*, *ResponseListener <Object> listener*)

Sets the current 3D status of the TV.

Related capabilities:

- `TVControl.3D.Set`

Parameters:

- `enabled` – Whether the TV's 3D mode should be on or off
- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

ServiceSubscription <State3DModeListener> **subscribe3DEnabled** (*State3DModeListener listener*)

Subscribes to changes in the TV's 3D status.

Related capabilities:

- `TVControl.3D.Subscribe`

Parameters:

- `listener` – (optional) `State3DModeListener` with methods to be called on success or failure

ToastControl **getToastControl** ()

CapabilityPriorityLevel <and-capabilityprioritylevel> **getToastControlCapabilityLevel** ()

void **showToast** (String *message*, *ResponseListener <Object> listener*)

Show a toast on the TV.

Parameters:

- message – Message to display
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **showClickableToastForApp** (String *message*, *AppInfo appInfo*, *JSONObject params*, *ResponseListener <Object> listener*)

Show a toast on the TV and perform an action when the toast is clicked on the TV.

Related capabilities:

- `ToastControl.Show.Clickable.App`
- `ToastControl.Show.Clickable.App.Params`

Parameters:

- message – Message to display
- appInfo – `AppInfo` for app to launch on click of toast
- params – Launch params for app
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **showClickableToastForURL** (String *message*, String *url*, *ResponseListener <Object> listener*)

Show a toast on the TV and perform an action when the toast is clicked on the TV.

Related capabilities:

- `ToastControl.Show.Clickable.URL`

Parameters:

- message – Message to display
- url
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

ExternalInputControl **getExternalInput** ()

CapabilityPriorityLevel **getExternalInputControlPriorityLevel** ()

void **launchInputPicker** (*AppLaunchListener listener*)

Launches the visual input picker on the device. This may be helpful for situations where the device does not support directly listing/modifying the external inputs.

Related capabilities:

- `ExternalInputControl.Picker.Launch`

Parameters:

- listener – (optional) `AppLaunchListener` with methods to be called on success or failure

void **closeInputPicker** (*LaunchSession launchSessionm*, *ResponseListener <Object> listener*)

Closes the input picker on the device, if it is currently open.

Related capabilities:

- `ExternalInputControl.Picker.Close`

Parameters:

- *launchSessionm*

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **getExternalInputList** (*ExternalInputListListener* listener)

Get a list of input devices (HDMI, AV, etc) connected to the device

Related capabilities:

- `ExternalInputControl.List`

Parameters:

- listener – (optional) `ExternalInputListListener` with methods to be called on success or failure

void **setExternalInput** (*ExternalInputInfo* input, *ResponseListener* <Object> listener)

Switch to the specified external input

Related capabilities:

- `ExternalInputControl.Set`

Parameters:

- input
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

MouseControl **getMouseControl** ()

CapabilityPriorityLevel **getMouseControlCapabilityLevel** ()

void **connectMouse** ()

Establish a connection with the `DeviceService`'s mouse communication medium (WebSocket, HTTP, etc). While this step may not be necessary with certain platforms, it is suggested to call it anyways, for purposes of seamless normalization. Calling `connect` on a non-connectable protocol will just trigger the success callback immediately.

Related capabilities:

- `MouseControl.Connect`

void **disconnectMouse** ()

Disconnects from the mouse communication medium.

Related capabilities:

- `MouseControl.Disconnect`

void **click** ()

Perform a click action at the current mouse position.

Related capabilities:

- `MouseControl.Click`

void **move** (double dx, double dy)

Move the mouse by the given distance values.

Related capabilities:

- `MouseControl.Move`

Parameters:

- dx – Distance to move the mouse on the x-axis relative to its current position

- `dy` – Distance to move the mouse on the y-axis relative to its current position

void **scroll** (double *dx*, double *dy*)

Scroll by the given distance values.

Related capabilities:

- `MouseControl.Scroll`

Parameters:

- `dx` – Distance to scroll the mouse on the x-axis relative to its current position
- `dy` – Distance to scroll the mouse on the y-axis relative to its current position

TextInputControl **getTextInputControl** ()

CapabilityPriorityLevel **getTextInputControlCapabilityLevel** ()

ServiceSubscription <*TextInputStatusListener*> **subscribeTextInputStatus** (*TextInputStatusListener* *listener*)

Subscribe to information about the current text field.

Related capabilities:

- `TextInputControl.Subscribe`

Parameters:

- *listener* – (optional) *TextInputStatusListener* with methods to be called on success or failure

void **sendText** (String *input*)

Send text to the current text field.

Related capabilities:

- `TextInputControl.Send.Text`

Parameters:

- *input*

void **sendEnter** () Send enter key to the current text field.

Related capabilities:

- `TextInputControl.Send.Enter`

void **sendDelete** ()

Send delete event to the current text field.

Related capabilities:

- `TextInputControl.Send.Delete`

PowerControl **getPowerControl** ()

CapabilityPriorityLevel **getPowerControlCapabilityLevel** ()

void **powerOff** (*ResponseListener* <Object> *listener*)

Sends a power off signal to the TV. A success message will, internally, trigger a disconnection with the device.

Related capabilities:

- `PowerControl.Off`

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **powerOn** (*ResponseListener* <Object> listener)

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

KeyControl **getKeyControl** ()

CapabilityPriorityLevel **getKeyControlCapabilityLevel** ()

void **up** (*ResponseListener* <Object> listener)

Sends the up button key code to the TV.

Related capabilities:

- `KeyControl.Up`

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **down** (*ResponseListener* <Object> listener)

Sends the down button key code to the TV.

Related capabilities:

- `KeyControl.Down`

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **left** (*ResponseListener* <Object> listener)

Sends the left button key code to the TV.

Related capabilities:

- `KeyControl.Left`

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **right** (*ResponseListener* <Object> listener)

Sends the right button key code to the TV.

Related capabilities:

- `KeyControl.Right`

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **ok** (*ResponseListener* <Object> listener)

Sends the OK button key code to the TV.

Related capabilities:

- `KeyControl.OK`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **back** (*ResponseListener* <Object> listener)

Sends the back button key code to the TV.

Related capabilities:

- `KeyControl.Back`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **home** (*ResponseListener* <Object> listener)

Sends the home button key code to the TV.

Related capabilities:

- `KeyControl.Home`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **sendKeyCode** (*KeyCode* keycode, *ResponseListener* <Object> listener)

Sends a key code value to the TV.

Related capabilities:

- `KeyControl.Send.KeyCode`

Parameters:

- keycode
- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

WebAppLauncher **getWebAppLauncher** ()

CapabilityPriorityLevel **getWebAppLauncherCapabilityLevel** ()

void **launchWebApp** (String *webAppId*, *LaunchListener* listener)

Launch a web application on the TV.

Related capabilities:

- `WebAppLauncher.Launch`
- `WebAppLauncher.Launch.Params` – if launching with params

Parameters:

- *webAppId* – ID of web app assigned by platform vendor
- listener – (optional) `LaunchListener` with methods to be called on success or failure

void **joinWebApp** (*LaunchSession* *webAppLaunchSession*, *LaunchListener* listener)

Join an active web app without launching/relaunching. If the app is not running/joinable, the failure block will be called immediately.

Related capabilities:

- `WebAppLauncher.Send`
- `WebAppLauncher.Receive`

Parameters:

- `webAppLaunchSession` – `LaunchSession` for the web app to be joined
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void **closeWebApp** (*LaunchSession* `launchSession`, *ResponseListener* `<Object> listener`)

Closes a web app with the provided `LaunchSession`.

Related capabilities:

- `WebAppLauncher.Close`

Parameters:

- `launchSession` – `LaunchSession` associated with the web app to be closed
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **pinWebApp** (String `webAppId`, *ResponseListener* `<Object> listener`)

Parameters:

- `webAppId`
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **unPinWebApp** (String `webAppId`, *ResponseListener* `<Object> listener`)

Parameters:

- `webAppId`
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **isWebAppPinned** (String `webAppId`, *WebAppPinStatusListener* `listener`)

Parameters:

- `webAppId`
- `listener` – (optional) `WebAppPinStatusListener` with methods to be called on success or failure

ServiceSubscription `<WebAppPinStatusListener>` **subscribeIsWebAppPinned** (String `webAppId`, *WebAppPinStatusListener* `listener`)

Parameters:

- `webAppId`
- `listener` – (optional) `WebAppPinStatusListener` with methods to be called on success or failure

PlaylistControl **getPlaylistControl** ()

CapabilityPriorityLevel **getPlaylistControlCapabilityLevel** ()

void **jumpToTrack** (long `index`, *ResponseListener* `<Object> listener`)

Jump the playlist to the designated track.

Play a track specified by index in the playlist

Related capabilities:

- `PlaylistControl.JumpToTrack`

Parameters:

- `index` – index in the playlist, it starts from zero like index of array

- listener – optional response listener

void **setPlayMode** (*PlayMode* playMode, *ResponseListener* <Object> listener)

Set order of playing tracks

Parameters:

- playMode
- listener – optional response listener

void **onLoseReachability** (*DeviceServiceReachability* reachability)

Parameters:

- reachability

void **unsubscribe** (*URLServiceSubscription*<?> subscription)

Parameters:

- subscription

void **sendCommand** (*ServiceCommand*<?> command)

Parameters:

- command

5.10.4 Capabilities

CapabilityPriorityLevel

`com.connectsdk.service.capability.CapabilityMethods.CapabilityPriorityLevel`

CapabilityPriorityLevel values are used by ConnectableDevice to find the most suitable DeviceService capability to be presented to the user. Values of VeryLow and VeryHigh are not in use internally the SDK. Connect SDK uses Low, Normal, and High internally.

Default behavior: If you are unsatisfied with the default priority levels & behavior of Connect SDK, it is possible to subclass a particular DeviceService and provide your own value for each capability. That DeviceService subclass would need to be registered with DiscoveryManager.

Properties

NOT_SUPPORTED = (0)

VERY_LOW = (1)

LOW = (25)

NORMAL = (50)

HIGH = (75)

VERY_HIGH = (100)

Methods

CapabilityPriorityLevel (int *value*) **Parameters:**

- value

int **getValue** ()

ExternalInputControl

com.connectsdk.service.capability.ExternalInputControl

extends CapabilityMethods

The ExternalInputControl capability serves to define the methods required for normalizing all functions regarding external input switching and general info.

Properties

final String Any = "ExternalInputControl.Any"

final String Picker_Launch = "ExternalInputControl.Picker.Launch"

final String Picker_Close = "ExternalInputControl.Picker.Close"

final String List = "ExternalInputControl.List"

final String Set = "ExternalInputControl.Set"

final String[] Capabilities = { Picker_Launch, Picker_Close, List, Set }

Inner Classes

- *ExternalInputListListener*

Methods

ExternalInputControl **getExternalInput** ()

CapabilityPriorityLevel **getExternalInputControlPriorityLevel** ()

void launchInputPicker (*AppLaunchListener listener*) Launches the visual input picker on the device. This may be helpful for situations where the device does not support directly listing/modifying the external inputs.

Related capabilities:

- ExternalInputControl.Picker.Launch

Parameters:

- listener – (optional) AppLaunchListener with methods to be called on success or failure

void closeInputPicker (*LaunchSession launchSessionm, ResponseListener <Object> listener*) Closes the input picker on the device, if it is currently open.

Related capabilities:

- ExternalInputControl.Picker.Close

Parameters:

- launchSessionm
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void `getExternalInputList` (*`ExternalInputListListener` listener*) Get a list of input devices (HDMI, AV, etc) connected to the device

Related capabilities:

- `ExternalInputControl.List`

Parameters:

- listener – (optional) `ExternalInputListListener` with methods to be called on success or failure

void `setExternalInput` (*`ExternalInputInfo` input, `ResponseListener <Object>` listener*) Switch to the specified external input

Related capabilities:

- `ExternalInputControl.Set`

Parameters:

- input
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

KeyControl

`com.connectsdk.service.capability.KeyControl`

extends `CapabilityMethods`

The `KeyControl` capability serves to define the methods required for normalizing common key commands (up, down, left right, ok, back, home, key code).

Properties

`final String Any = "KeyControl.Any"`

`final String Up = "KeyControl.Up"`

`final String Down = "KeyControl.Down"`

`final String Left = "KeyControl.Left"`

`final String Right = "KeyControl.Right"`

`final String OK = "KeyControl.OK"`

`final String Back = "KeyControl.Back"`

`final String Home = "KeyControl.Home"`

`final String Send_Key = "KeyControl.SendKey"`

`final String KeyCode = "KeyControl.KeyCode"`

`final String[] Capabilities = { Up, Down, Left, Right, OK, Back, Home,**`

Inner Classes

- *KeyCode*

Methods

KeyControl **getKeyControl** ()

CapabilityPriorityLevel **getKeyControlCapabilityLevel** ()

void up (*ResponseListener* <Object> *listener*) Sends the up button key code to the TV.

Related capabilities:

- *KeyControl.Up*

Parameters:

- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void down (*ResponseListener* <Object> *listener*) Sends the down button key code to the TV.

Related capabilities:

- *KeyControl.Down*

Parameters:

- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void left (*ResponseListener* <Object> *listener*) Sends the left button key code to the TV.

Related capabilities:

- *KeyControl.Left*

Parameters:

- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void right (*ResponseListener* <Object> *listener*) Sends the right button key code to the TV.

Related capabilities:

- *KeyControl.Right*

Parameters:

- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void ok (*ResponseListener* <Object> *listener*) Sends the OK button key code to the TV.

Related capabilities:

- *KeyControl.OK*

Parameters:

- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void back (*ResponseListener* <Object> *listener*) Sends the back button key code to the TV.

Related capabilities:

- *KeyControl.Back*

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void home (*`ResponseListener <Object> listener`*) Sends the home button key code to the TV.

Related capabilities:

- `KeyControl.Home`

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void sendKeyCode (*`KeyCode keycode`, `ResponseListener <Object> listener`*) Sends a key code value to the TV.

Related capabilities:

- `KeyControl.Send.KeyCode`

Parameters:

- keycode
- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

Launcher

`com.connectsdk.service.capability.Launcher`

extends `CapabilityMethods`

The Launcher capability protocol serves to define the methods required for normalizing the launching of apps. It allows for in-built support for certain common launch types (deep-linking to YouTube, Netflix, Hulu, browser, etc) as well as by (platform-specific) app id.

Properties

```
final String Any = "Launcher.Any"
final String Application = "Launcher.App"
final String Application_Params = "Launcher.App.Params"
final String Application_Close = "Launcher.App.Close"
final String Application_List = "Launcher.App.List"
final String Browser = "Launcher.Browser"
final String Browser_Params = "Launcher.Browser.Params"
final String Hulu = "Launcher.Hulu"
final String Hulu_Params = "Launcher.Hulu.Params"
final String Netflix = "Launcher.Netflix"
final String Netflix_Params = "Launcher.Netflix.Params"
final String YouTube = "Launcher.YouTube"
final String YouTube_Params = "Launcher.YouTube.Params"
final String AppStore = "Launcher.AppStore"
final String AppStore_Params = "Launcher.AppStore.Params"
```

```
final String AppState = "Launcher.AppState"
```

```
final String AppState_Subscribe = "Launcher.AppState.Subscribe"
```

```
final String RunningApp = "Launcher.RunningApp"
```

```
final String RunningApp_Subscribe = "Launcher.RunningApp.Subscribe"
```

```
final String[] Capabilities = { Application, Application_Params, Application_Close, Application_List, Browser, Browser_Params, Hulu, Hulu_Params, Netflix, Netflix_Params, YouTube, YouTube_Params, AppStore, AppStore_Params, AppState, AppState_Subscribe, RunningApp, RunningApp_Subscribe }
```

Inner Classes

- *AppInfoListener*
- *AppLaunchListener*
- *AppListListener*
- *AppState*
- *AppStateListener*

Methods

Launcher **getLauncher** ()

CapabilityPriorityLevel **getLauncherCapabilityLevel** ()

void launchAppWithInfo (*AppInfo appInfo, AppLaunchListener listener*) Launch an application on the device.

Related capabilities:

- `Launcher.App`
- `Launcher.App.Params` – if launching with params

Parameters:

- `appInfo` – *AppInfo* object for the application
- `listener` – (optional) *AppLaunchListener* with methods to be called on success or failure

void launchAppWithInfo (*AppInfo appInfo, Object params, AppLaunchListener listener*) Launch an application on the device.

Related capabilities:

- `Launcher.App`
- `Launcher.App.Params` – if launching with params

Parameters:

- `appInfo` – *AppInfo* object for the application
- `params`
- `listener` – (optional) *AppLaunchListener* with methods to be called on success or failure

void launchApp (`String appId, AppLaunchListener listener`) Launch an application on the device.

Related capabilities:

- `Launcher.App`

Parameters:

- `appId` – ID of the application
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void closeApp (*LaunchSession launchSession, ResponseListener <Object> listener*) Close an application on the device.

Related capabilities:

- `Launcher.App.Close`

Parameters:

- `launchSession` – `LaunchSession` of the target app
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getAppList (*AppListListener listener*) Gets a list of all apps installed on the device.

Related capabilities:

- `Launcher.App.List`

Parameters:

- `listener` – (optional) `AppListListener` with methods to be called on success or failure

void getRunningApp (*AppInfoListener listener*) Gets an `AppInfo` object for the current running app on the device.

Related capabilities:

- `Launcher.RunningApp`

Parameters:

- `listener` – (optional) `AppInfoListener` with methods to be called on success or failure

ServiceSubscription <AppInfoListener> **subscribeRunningApp** (*AppInfoListener listener*) Subscribes to changes of the current running app. Every time the running app changes, the success block will be called with an `AppInfo` object for the current running app.

Related capabilities:

- `Launcher.RunningApp.Subscribe`

Parameters:

- `listener` – (optional) `AppInfoListener` with methods to be called on success or failure

void getAppState (*LaunchSession launchSession, AppStateListener listener*) Gets the target app's running status and on-screen visibility.

Related capabilities:

- `Launcher.AppState`

Parameters:

- `launchSession` – `LaunchSession` of the target app
- `listener` – (optional) `AppStateListener` with methods to be called on success or failure

ServiceSubscription <AppStateListener> **subscribeAppState** (*LaunchSession launchSession, AppStateListener listener*) Subscribes to changes of the state of the target app. Every time the app's state changes, the success block will be called with info on the app's running status and on-screen visibility.

Related capabilities:

- `Launcher.AppState.Subscribe`

Parameters:

- `launchSession` - `LaunchSession` of the target app
- `listener` – (optional) `AppStateListener` with methods to be called on success or failure

void launchBrowser (String *url*, *AppLaunchListener listener*) Launch the web browser. Will launch deep-linked to provided URL, if supported on the target platform.

Related capabilities:

- `Launcher.Browser`
- `Launcher.Browser.Params` – if launching with url

Parameters:

- `url`
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void launchYouTube (String *contentId*, *AppLaunchListener listener*) Launch YouTube app. Will launch deep-linked to provided contentId, if supported on the target platform.

Related capabilities:

- `Launcher.YouTube`
- `Launcher.YouTube.Params` – if launching with contentId

Parameters:

- `contentId` – Video id to open
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void launchYouTube (String *contentId*, float *startTime*, *AppLaunchListener listener*) Launch YouTube app. Will launch deep-linked to provided contentId, if supported on the target platform.

Related capabilities:

- `Launcher.YouTube`
- `Launcher.YouTube.Params` – if launching with contentId

Parameters:

- `contentId` – Video id to open
- `startTime`
- `listener` – (optional) `AppLaunchListener` with methods to be called on success or failure

void launchNetflix (String *contentId*, *AppLaunchListener listener*) Launch Netflix app. Will launch deep-linked to provided contentId, if supported on the target platform.

Related capabilities:

- `Launcher.Netflix`
- `Launcher.Netflix.Params` – if launching with contentId

Parameters:

- `contentId` – Video id to open

- listener – (optional) `AppLaunchListener` with methods to be called on success or failure

void launchHulu (String *contentId*, *AppLaunchListener* listener) Launch Hulu app. Will launch deep-linked to provided *contentId*, if supported on the target platform.

Related capabilities:

- `Launcher.Hulu`
- `Launcher.Hulu.Params` – if launching with *contentId*

Parameters:

- *contentId* – Video id to open
- listener – (optional) `AppLaunchListener` with methods to be called on success or failure

void launchAppStore (String *appId*, *AppLaunchListener* listener) Launch the device’s app store app, optionally deep-linked to a specific app’s page.

Related capabilities:

- `Launcher.AppStore`
- `Launcher.AppStore.Params`

Parameters:

- *appId* – (optional) ID of the application to show in the app store
- listener – (optional) `AppLaunchListener` with methods to be called on success or failure

MediaControl

`com.connectsdk.service.capability.MediaControl`

extends CapabilityMethods

The `MediaControl` capability protocol serves to define the methods required for normalizing the control of media playback (play, pause, fast forward, etc) as well as obtaining media information (playhead position, duration, etc).

Properties

`final String Any = “MediaControl.Any”`

`final String Play = “MediaControl.Play”`

`final String Pause = “MediaControl.Pause”`

`final String Stop = “MediaControl.Stop”`

`final String Rewind = “MediaControl.Rewind”`

`final String FastForward = “MediaControl.FastForward”`

`final String Seek = “MediaControl.Seek”`

`final String Duration = “MediaControl.Duration”`

`final String PlayState = “MediaControl.PlayState”`

`final String PlayState_Subscribe = “MediaControl.PlayState.Subscribe”`

`final String Position = “MediaControl.Position”`

final String Previous = “MediaControl.Previous” This capability is deprecated. Use `PlaylistControl.Previous` instead.

final String Next = “MediaControl.Next” This capability is deprecated. Use `PlaylistControl.Next` instead.

final int PLAYER_STATE_UNKNOWN = 0

final int PLAYER_STATE_IDLE = 1

final int PLAYER_STATE_PLAYING = 2

final int PLAYER_STATE_PAUSED = 3

final int PLAYER_STATE_BUFFERING = 4

final String[] Capabilities = { Play, Pause, Stop, Rewind, FastForward, Seek,

Inner Classes

- *DurationListener*
- *PlayStateListener*
- *PlayStateStatus*
- *PositionListener*

Methods

MediaControl getMediaControl () Get MediaControl implementation

Returns: MediaControl

CapabilityPriorityLevel getMediaControlCapabilityLevel () Get a capability priority for current implementation

Returns: CapabilityPriorityLevel

void play (ResponseListener <Object> listener) Send play command.

Related capabilities:

- `MediaControl.Play`

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void pause (ResponseListener <Object> listener) Send pause command.

Related capabilities:

- `MediaControl.Pause`

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void stop (ResponseListener <Object> listener) Send stop command.

Related capabilities:

- `MediaControl.Stop`

Parameters:

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void rewind (*ResponseListener* <Object> *listener*) Send rewind command.

Related capabilities:

- `MediaControl.Rewind`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void fastForward (*ResponseListener* <Object> *listener*) Send play command.

Related capabilities:

- `MediaControl.FastForward`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void previous (*ResponseListener* <Object> *listener*) This method is deprecated. Use `PlaylistControl::previous (ResponseListener<Object> listener)` instead.

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void next (*ResponseListener* <Object> *listener*) This method is deprecated. Use `PlaylistControl::next (ResponseListener<Object> listener)` instead.

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void seek (long *position*, *ResponseListener* <Object> *listener*) Seeks to a new position within the current media item

Related capabilities:

- `MediaControl.Seek`

Parameters:

- *position* – The new position, in milliseconds from the beginning of the stream
- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void getDuration (*DurationListener* *listener*) Get the current media duration in milliseconds

Parameters:

- *listener* – (optional) `DurationListener` with methods to be called on success or failure

void getPosition (*PositionListener* *listener*) Get the current playback position in milliseconds

Parameters:

- *listener* – (optional) `PositionListener` with methods to be called on success or failure

void getPlayState (*PlayStateListener* *listener*) Get the current state of playback

Parameters:

- *listener* – (optional) `PlayStateListener` with methods to be called on success or failure

ServiceSubscription <*PlayStateListener*> **subscribePlayState** (*PlayStateListener* *listener*) Subscribe for playback state changes

Parameters:

- listener – receives play state notifications

Returns: ServiceSubscription<PlayStateListener>

MediaPlayer

com.connectsdk.service.capability.MediaPlayer

extends CapabilityMethods

The MediaPlayer capability protocol serves to define the methods required for displaying media on the device.

Properties

final String Any = “MediaPlayer.Any”

final String Display_Video = “MediaPlayer.Play.Video” This capability is deprecated. Use MediaPlayer.Play_Video instead.

final String Display_Audio = “MediaPlayer.Play.Audio” This capability is deprecated. Use MediaPlayer.Play_Audio instead.

final String Display_Image = “MediaPlayer.Display.Image”

final String Play_Video = “MediaPlayer.Play.Video”

final String Play_Audio = “MediaPlayer.Play.Audio”

final String Play_Playlist = “MediaPlayer.Play.Playlist”

final String Close = “MediaPlayer.Close”

final String Loop = “MediaPlayer.Loop”

final String Subtitle_SRT = “MediaPlayer.Subtitle.SRT”

final String Subtitle_WebVTT = “MediaPlayer.Subtitle.WebVTT”

final String MetaData_Title = “MediaPlayer.MetaData.Title”

final String MetaData_Description = “MediaPlayer.MetaData.Description”

final String MetaData_Thumbnail = “MediaPlayer.MetaData.Thumbnail”

final String MetaData_MimeType = “MediaPlayer.MetaData.MimeType”

final String MediaInfo_Get = “MediaPlayer.MediaInfo.Get”

final String MediaInfo_Subscribe = “MediaPlayer.MediaInfo.Subscribe”

final String[] Capabilities = { Display_Image, Play_Video, Play_Audio, Close, MetaData_Title, MetaData_Description, MetaData_Thumbnail, MetaData_MimeType, MediaInfo_Get, MediaInfo_Subscribe }

Inner Classes

- *LaunchListener*
- *MediaInfoListener*
- *MediaLaunchObject*

Methods

MediaPlayer **getMediaPlayer** ()

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel** ()

void getMediaInfo (*MediaInfoListener* listener) **Parameters:**

- listener – (optional) *MediaInfoListener* with methods to be called on success or failure

ServiceSubscription <*MediaInfoListener*> **subscribeMediaInfo** (*MediaInfoListener* listener) **Parameters:**

- listener – (optional) *MediaInfoListener* with methods to be called on success or failure

void displayImage (*MediaInfo* mediaInfo, *LaunchListener* listener) Display an image on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- *MediaPlayer.Display.Image*
- *MediaPlayer.MediaData.Title*
- *MediaPlayer.MediaData.Description*
- *MediaPlayer.MediaData.Thumbnail*
- *MediaPlayer.MediaData.MimeType*

Parameters:

- mediaInfo – Object of *MediaInfo* class which includes all the information about an image to display.
- listener – (optional) *LaunchListener* with methods to be called on success or failure

void playMedia (*MediaInfo* mediaInfo, *boolean* shouldLoop, *LaunchListener* listener) Play an audio or video file on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- *MediaPlayer.Play.Video*
- *MediaPlayer.Play.Audio*
- *MediaPlayer.MediaData.Title*
- *MediaPlayer.MediaData.Description*
- *MediaPlayer.MediaData.Thumbnail*
- *MediaPlayer.MediaData.MimeType*

Parameters:

- mediaInfo – Object of *MediaInfo* class which includes all the information about an image to display.
- shouldLoop – Whether to automatically loop playback
- listener – (optional) *LaunchListener* with methods to be called on success or failure

void closeMedia (*LaunchSession* launchSession, *ResponseListener* <*Object*> listener) Close a running media session. Because media is handled differently on different platforms, it is required to keep track of *LaunchSession* and *MediaControl* objects to control that media session in the future. *LaunchSession* will be required to close the media and *mediaControl* will be required to control the media.

Related capabilities:

- *MediaPlayer.Close*

Parameters:

- `launchSession` – `LaunchSession` object for use in closing media instance
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void `displayImage` (String *url*, String *mimeType*, String *title*, String *description*, String *iconSrc*, `LaunchListener` *listener*)

Display an image on the device. Not all devices support all of the parameters – supply as many as you have available.

This method is deprecated. Use `MediaPlayer::displayImage(MediaInfo mediaInfo, LaunchListener listener)` instead.

Related capabilities:

- `MediaPlayer.Display.Image`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- `url`
- `mimeType` – MIME type of the image, for example “image/jpeg”
- `title` – Title text to display
- `description` – Description text to display
- `iconSrc`
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void `playMedia` (String *url*, String *mimeType*, String *title*, String *description*, String *iconSrc*, boolean *shouldLoop*, `LaunchListener` *listener*)

Play an audio or video file on the device. Not all devices support all of the parameters – supply as many as you have available.

This method is deprecated. Use `MediaPlayer::playMedia(MediaInfo mediaInfo, boolean shouldLoop, LaunchListener listener)` instead.

Related capabilities:

- `MediaPlayer.Play.Video`
- `MediaPlayer.Play.Audio`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- `url`
- `mimeType` – MIME type of the video, for example “video/mpeg4”, “audio/mp3”, etc
- `title` – Title text to display
- `description` – Description text to display

- iconSrc
- shouldLoop – Whether to automatically loop playback
- listener – (optional) `LaunchListener` with methods to be called on success or failure

MouseControl

`com.connectsdk.service.capability.MouseControl`

extends CapabilityMethods

The `MouseControl` capability serves to define the methods required for normalizing a mouse/trackpad (move/scroll with relative coordinates and click).

Properties

`final String Any = "MouseControl.Any"`

`final String Connect = "MouseControl.Connect"`

`final String Disconnect = "MouseControl.Disconnect"`

`final String Click = "MouseControl.Click"`

`final String Move = "MouseControl.Move"`

`final String Scroll = "MouseControl.Scroll"`

`final String[] Capabilities = { Connect, Disconnect, Click, Move, Scroll }`

Methods

MouseControl `getMouseControl ()`

CapabilityPriorityLevel `getMouseControlCapabilityLevel ()`

void connectMouse () Establish a connection with the `DeviceService`'s mouse communication medium (WebSocket, HTTP, etc). While this step may not be necessary with certain platforms, it is suggested to call it anyways, for purposes of seamless normalization. Calling `connect` on a non-connectable protocol will just trigger the success callback immediately.

Related capabilities:

- `MouseControl.Connect`

void disconnectMouse () Disconnects from the mouse communication medium.

Related capabilities:

- `MouseControl.Disconnect`

void click () Perform a click action at the current mouse position.

Related capabilities:

- `MouseControl.Click`

void move (double dx, double dy) Move the mouse by the given distance values.

Related capabilities:

- `MouseControl.Move`

Parameters:

- dx – Distance to move the mouse on the x-axis relative to its current position
- dy – Distance to move the mouse on the y-axis relative to its current position

void move (PointF *distance*) Move the mouse by the given distance values.

Related capabilities:

- `MouseControl.Move`

Parameters:

- distance – Distance to move the mouse relative to its current position

void scroll (double *dx*, double *dy*) Scroll by the given distance values.

Related capabilities:

- `MouseControl.Scroll`

Parameters:

- dx – Distance to scroll the mouse on the x-axis relative to its current position
- dy – Distance to scroll the mouse on the y-axis relative to its current position

void scroll (PointF *distance*) Scroll by the given distance values.

Related capabilities:

- `MouseControl.Scroll`

Parameters:

- distance – Distance to scroll relative to current position

PlaylistControl

```
com.connectsdk.service.capability.PlaylistControl
```

extends CapabilityMethods

The PlaylistControl capability interface serves to define the methods required for normalizing the control of playlist (next, previous, jumpToTrack, etc)

Properties

```
final String Any = "PlaylistControl.Any"
```

```
final String JumpToTrack = "PlaylistControl.JumpToTrack"
```

```
final String SetPlayMode = "PlaylistControl.SetPlayMode"
```

```
final String Previous = "PlaylistControl.Previous"
```

```
final String Next = "PlaylistControl.Next"
```

```
final String[] Capabilities = { Previous, Next, JumpToTrack, SetPlayMode, JumpToTrack, }
```

Inner Classes

- *PlayMode*

Methods

PlaylistControl **getPlaylistControl** ()

CapabilityPriorityLevel **getPlaylistControlCapabilityLevel** ()

void previous (*ResponseListener* <Object> *listener*) Jump playlist to the previous track.

Play previous track in the playlist

Related capabilities:

- `PlaylistControl.Previous`

Parameters:

- *listener* – optional response listener

void next (*ResponseListener* <Object> *listener*) Jump playlist to the next track.

Play next track in the playlist

Related capabilities:

- `PlaylistControl.Next`

Parameters:

- *listener* – optional response listener

void jumpToTrack (**long** *index*, *ResponseListener* <Object> *listener*) Jump the playlist to the designated track.

Play a track specified by index in the playlist

Related capabilities:

- `PlaylistControl.JumpToTrack`

Parameters:

- *index* – index in the playlist, it starts from zero like index of array
- *listener* – optional response listener

void setPlayMode (*PlayMode* *playMode*, *ResponseListener* <Object> *listener*) Set order of playing tracks

Parameters:

- *playMode*
- *listener* – optional response listener

PowerControl

`com.connectsdk.service.capability.PowerControl`

extends `CapabilityMethods`

The `PowerControl` capability protocol serves to define the methods required for normalizing power off functionality.

Properties

```
final String Any = "PowerControl.Any"  
final String Off = "PowerControl.Off"  
final String On = "PowerControl.On"  
final String[] Capabilities = { Off, On }
```

Methods

PowerControl **getPowerControl ()**

CapabilityPriorityLevel **getPowerControlCapabilityLevel ()**

void powerOff (*ResponseListener* <Object> *listener*) Sends a power off signal to the TV. A success message will, internally, trigger a disconnection with the device.

Related capabilities:

- `PowerControl.Off`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void powerOn (*ResponseListener* <Object> *listener*) **Parameters:**

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

TVControl

```
com.connectsdk.service.capability.TVControl
```

extends CapabilityMethods

The TVControl capability protocol serves to define the methods required for normalizing common TV-specific commands (channel up/down, channel list, channel info, etc).

Properties

```
final String Any = "TVControl.Any"  
final String Channel_Get = "TVControl.Channel.Get"  
final String Channel_Set = "TVControl.Channel.Set"  
final String Channel_Up = "TVControl.Channel.Up"  
final String Channel_Down = "TVControl.Channel.Down"  
final String Channel_List = "TVControl.Channel.List"  
final String Channel_Subscribe = "TVControl.Channel.Subscribe"  
final String Program_Get = "TVControl.Program.Get"  
final String Program_List = "TVControl.Program.List"  
final String Program_Subscribe = "TVControl.Program.Subscribe"
```

```

final String Program_List_Subscribe = "TVControl.Program.List.Subscribe"
final String Get_3D = "TVControl.3D.Get"
final String Set_3D = "TVControl.3D.Set"
final String Subscribe_3D = "TVControl.3D.Subscribe"
final String[] Capabilities = { Channel_Get, Channel_Set, Channel_Up, Channel_Down, Channel_List, Channel_Subscribe, Program_Get, Program_List, Program_Subscribe, Program_List_Subscribe, Get_3D, Set_3D, Subscribe_3D }

```

Inner Classes

- *ChannelListener*
- *ChannelListListener*
- *ProgramInfoListener*
- *ProgramListListener*
- *State3DModeListener*

Methods

TVControl **getTVControl** ()

CapabilityPriorityLevel **getTVControlCapabilityLevel** ()

void **channelUp** (*ResponseListener*<Object> *listener*)

Sends a channel up command to the TV.

Related capabilities:

- `TVControl.Channel.Up`

Parameters:

- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void **channelDown** (*ResponseListener* <Object> *listener*)

Sends a channel down command to the TV.

Related capabilities:

- `TVControl.Channel.Down`

Parameters:

- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void **setChannel** (*ChannelInfo* *channelNumber*, *ResponseListener* <Object> *listener*)

Sets the current channel to the channel provided by the *ChannelInfo* object provided.

Related capabilities:

- `TVControl.Channel.Set`

Parameters:

- *channelNumber*

- listener – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **getCurrentChannel** (*ChannelListener* listener)

Gets the current channel info from the TV.

Related capabilities:

- `TVControl.Channel.Get`

Parameters:

- listener – (optional) `ChannelListener` with methods to be called on success or failure

ServiceSubscription <ChannelListener> **subscribeCurrentChannel** (*ChannelListener <and-channellistener>* listener)

Subscribes to any changes in the current channel. Each time the channel is changed, the new channel's info will be provided to the success callback.

Related capabilities:

- `TVControl.Channel.Subscribe`

Parameters:

- listener – (optional) `ChannelListener` with methods to be called on success or failure

void **getChannelList** (*ChannelListListener* listener)

Get a list of available channels from the TV.

Related capabilities:

- `TVControl.Channel.List`

Parameters:

- listener – (optional) `ChannelListListener` with methods to be called on success or failure

void **getProgramInfo** (*ProgramInfoListener* listener)

Gets the current program info from the TV.

Related capabilities:

- `TVControl.Program.Get`

Parameters:

- listener – (optional) `ProgramInfoListener` with methods to be called on success or failure

ServiceSubscription <ProgramInfoListener> **subscribeProgramInfo** (*ProgramInfoListener* listener)

Subscribes to any changes in the current program. Each time the channel is changed or a new program starts, the new program's info will be provided to the success callback.

Related capabilities:

- `TVControl.Program.Subscribe`

Parameters:

- listener – (optional) `ProgramInfoListener` with methods to be called on success or failure

void **getProgramList** (*ProgramListListener* listener)

Gets a list of all programs scheduled to play on the current channel.

Related capabilities:

- `TVControl.Program.List`

Parameters:

- listener – (optional) `ProgramListListener` with methods to be called on success or failure

ServiceSubscription `<ProgramListListener> subscribeProgramList (ProgramListListener listener)`

Subscribes to any changes in the current program. Each time the channel is changed or a new program starts, the new program's info will be provided to the success callback.

Related capabilities:

- `TVControl.Program.List.Subscribe`

Parameters:

- listener – (optional) `ProgramListListener` with methods to be called on success or failure

void **get3DEnabled** (*State3DModeListener* listener)

Gets the current 3D status of the TV.

Related capabilities:

- `TVControl.3D.Get`

Parameters:

- listener – (optional) `State3DModeListener` with methods to be called on success or failure

void **set3DEnabled** (boolean *enabled*, *ResponseListener* `<Object>` listener)

Sets the current 3D status of the TV.

Related capabilities:

- `TVControl.3D.Set`

Parameters:

- *enabled* – Whether the TV's 3D mode should be on or off
- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

ServiceSubscription `<State3DModeListener> subscribe3DEnabled (State3DModeListener listener)`

Subscribes to changes in the TV's 3D status.

Related capabilities:

- `TVControl.3D.Subscribe`

Parameters:

- listener – (optional) `State3DModeListener` with methods to be called on success or failure

TextInputControl

`com.connectsdk.service.capability.TextInputControl`

extends `CapabilityMethods`

The `TextInputControl` capability serves to define the methods required for normalizing common text input commands (send text, enter, delete, keyboard status).

Properties

```
final String Any = "TextInputControl.Any"  
final String Send = "TextInputControl.Send"  
final String Send_Enter = "TextInputControl.Enter"  
final String Send_Delete = "TextInputControl.Delete"  
final String Subscribe = "TextInputControl.Subscribe"  
final String[] Capabilities = { Send, Send_Enter, Send_Delete, Subscribe }
```

Inner Classes

- *TextInputStatusListener*

Methods

TextInputControl **getTextInputControl** ()

CapabilityPriorityLevel **getTextInputControlCapabilityLevel** ()

ServiceSubscription < *TextInputStatusListener* > **subscribeTextInputStatus** (*TextInputStatusListener* listener)

Subscribe to information about the current text field.

Related capabilities:

- `TextInputControl.Subscribe`

Parameters:

- listener – (optional) *TextInputStatusListener* with methods to be called on success or failure

void **sendText** (String *input*)

Send text to the current text field.

Related capabilities:

- `TextInputControl.Send.Text`

Parameters:

- *input*

void **sendEnter** ()

Send enter key to the current text field.

Related capabilities:

- `TextInputControl.Send.Enter`

void **sendDelete** ()

Send delete event to the current text field.

Related capabilities:

- `TextInputControl.Send.Delete`

ToastControl

com.connectsdk.service.capability.ToastControl

extends CapabilityMethods

The ToastControl capability protocol serves to define the methods required for displaying toast messages on the TV.

Toasts may optionally provide an 80x80 pixel icon in PNG or JPEG format, encoded as base64. The icon will be displayed alongside the toast message.

Properties

final String Any = “ToastControl.Any”

final String Show_Toast = “ToastControl.Show”

final String Show_Clickable_Toast_App = “ToastControl.Show.Clickable.App”

final String Show_Clickable_Toast_App_Params = “ToastControl.Show.Clickable.App.Params”

final String Show_Clickable_Toast_URL = “ToastControl.Show.Clickable.URL”

final String[] Capabilities = { Show_Toast, Show_Clickable_Toast_App, Show_Clickable_Toast_App_Params, Show_Clickable_Toast_URL }

Methods

ToastControl **getToastControl** ()

CapabilityPriorityLevel **getToastControlCapabilityLevel** ()

void **showToast** (String *message*, *ResponseListener*<Object> *listener*)

Show a toast on the TV.

Parameters:

- *message* – Message to display
- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void **showToast** (String *message*, String *iconData*, String *iconExtension*, *ResponseListener* <Object> *listener*)

Show a toast on the TV.

Parameters:

- *message* – Message to display
- *iconData* – Base-64 encoded JPEG or PNG data
- *iconExtension* – File extension of icon
- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

void **showClickableToastForApp** (String *message*, *AppInfo* *appInfo*, *JSONObject* *params*, *ResponseListener* <Object> *listener*)

Show a toast on the TV and perform an action when the toast is clicked on the TV.

Related capabilities:

- `ToastControl.Show.Clickable.App`

- `ToastControl.Show.Clickable.App.Params`

Parameters:

- `message` – Message to display
- `appInfo` – `AppInfo` for app to launch on click of toast
- `params` – Launch params for app
- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **showClickableToastForApp** (String *message*, *AppInfo* *appInfo*, *JSONObject* *params*, String *iconData*, String *iconExtension*, *ResponseListener* <Object> *listener*)

Show a toast on the TV and perform an action when the toast is clicked on the TV.

Related capabilities:

- `ToastControl.Show.Clickable.App`
- `ToastControl.Show.Clickable.App.Params`

Parameters:

- `message` – Message to display
- `appInfo` – `AppInfo` for app to launch on click of toast
- `params` – Launch params for app
- `iconData` – Base-64 encoded JPEG or PNG data
- `iconExtension` – File extension of icon
- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **showClickableToastForURL** (String *message*, String *url*, *ResponseListener* <Object> *listener*)

Show a toast on the TV and perform an action when the toast is clicked on the TV.

Related capabilities:

- `ToastControl.Show.Clickable.URL`

Parameters:

- `message` – Message to display
- `url`
- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **showClickableToastForURL** (String *message*, String *url*, String *iconData*, String *iconExtension*, *ResponseListener* <Object> *listener*)

Show a toast on the TV and perform an action when the toast is clicked on the TV.

Related capabilities:

- `ToastControl.Show.Clickable.URL`

Parameters:

- `message` – Message to display
- `url`
- `iconData` – Base-64 encoded JPEG or PNG data
- `iconExtension` – File extension of icon

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

VolumeControl

`com.connectsdk.service.capability.VolumeControl`

extends CapabilityMethods

The VolumeControl capability protocol serves to define the methods required for normalizing common volume specific commands (volume up/down, mute, etc).

Properties

final String Any = “VolumeControl.Any”

final String Volume_Get = “VolumeControl.Get”

final String Volume_Set = “VolumeControl.Set”

final String Volume_Up_Down = “VolumeControl.UpDown”

final String Volume_Subscribe = “VolumeControl.Subscribe”

final String Mute_Get = “VolumeControl.Mute.Get”

final String Mute_Set = “VolumeControl.Mute.Set”

final String Mute_Subscribe = “VolumeControl.Mute.Subscribe”

final String[] Capabilities = { Volume_Get, Volume_Set, Volume_Up_Down, Volume_Subscribe, Mute_Get, Mute_Set, Mute_Subscribe }

Inner Classes

- *MuteListener*
- *VolumeListener*
- *VolumeStatus*
- *VolumeStatusListener*

Methods

VolumeControl **getVolumeControl** ()

CapabilityPriorityLevel **getVolumeControlCapabilityLevel** ()

void **volumeUp** (*ResponseListener<Object>* listener)

Sends the volume up command to the device.

Related capabilities:

- `VolumeControl.UpDown`

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **volumeDown** (*ResponseListener <Object>* listener)

Sends the volume down command to the device.

Related capabilities:

- `VolumeControl.UpDown`

Parameters:

- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **setVolume** (float *volume*, *ResponseListener* <Object> *listener*)

Set the volume of the device.

Related capabilities:

- `VolumeControl.Set`

Parameters:

- `volume` – Volume as a float between 0.0 and 1.0
- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **getVolume** (*VolumeListener* *listener*)

Get the current volume of the device.

Related capabilities:

- `VolumeControl.Get`

Parameters:

- `listener` – (optional) `VolumeListener` with methods to be called on success or failure

void **setMute** (boolean *isMute*, *ResponseListener* <Object> *listener*)

Set the current volume.

Related capabilities:

- `VolumeControl.Mute.Set`

Parameters:

- `isMute`
- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **getMute** (*MuteListener* *listener*)

Get the current mute state.

Related capabilities:

- `VolumeControl.Mute.Get`

Parameters:

- `listener` – (optional) `MuteListener` with methods to be called on success or failure

ServiceSubscription <*VolumeListener*> **subscribeVolume** (*VolumeListener* *listener*)

Subscribe to the volume on the TV.

Related capabilities:

- `VolumeControl.Subscribe`

Parameters:

- listener – (optional) VolumeListener with methods to be called on success or failure

ServiceSubscription <*MuteListener*> **subscribeMute** (*MuteListener* listener)

Subscribe to the mute state on the TV.

Related capabilities:

- `VolumeControl.Mute.Subscribe`

Parameters:

- listener – (optional) MuteListener with methods to be called on success or failure

WebAppLauncher

`com.connectsdk.service.capability.WebAppLauncher`

extends CapabilityMethods

The WebAppLauncher capability protocol provides capabilities for launching web apps and establishing two-way communication.

Properties

final String Any = “WebAppLauncher.Any”

final String Launch = “WebAppLauncher.Launch”

final String Launch_Params = “WebAppLauncher.Launch.Params”

final String Message_Send = “WebAppLauncher.Message.Send”

final String Message_Receive = “WebAppLauncher.Message.Receive”

final String Message_Send_JSON = “WebAppLauncher.Message.Send.JSON”

final String Message_Receive_JSON = “WebAppLauncher.Message.Receive.JSON”

final String Connect = “WebAppLauncher.Connect”

final String Disconnect = “WebAppLauncher.Disconnect”

final String Join = “WebAppLauncher.Join”

final String Close = “WebAppLauncher.Close”

final String Pin = “WebAppLauncher.Pin”

final String[] Capabilities = { **Launch**, **Launch_Params**, **Message_Send**, **Message_Receive**, **Message_Send_JSON**, **Message_Receive_JSON**, **Connect**, **Disconnect**, **Join**, **Close**, **Pin** }

Methods

WebAppLauncher **getWebAppLauncher** ()

CapabilityPriorityLevel **getWebAppLauncherCapabilityLevel** ()

void **launchWebApp** (String *webAppId*, LaunchListener *listener*)

Launch a web application on the TV.

Related capabilities:

- `WebAppLauncher.Launch`
- `WebAppLauncher.Launch.Params` – if launching with params

Parameters:

- `webAppId` – ID of web app assigned by platform vendor
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void **launchWebApp** (String *webAppId*, boolean *relaunchIfRunning*, `LaunchListener` *listener*)

Launch a web application on the TV.

Related capabilities:

- `WebAppLauncher.Launch`
- `WebAppLauncher.Launch.Params` – if launching with params

Parameters:

- `webAppId` – ID of web app assigned by platform vendor
- `relaunchIfRunning` – If supported on target platform, web app will force relaunch if value true
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void **launchWebApp** (String *webAppId*, `JSONObject` *params*, `LaunchListener` *listener*)

Launch a web application on the TV.

Related capabilities:

- `WebAppLauncher.Launch`
- `WebAppLauncher.Launch.Params` – if launching with params

Parameters:

- `webAppId` – ID of web app assigned by platform vendor
- `params` – Dictionary of key/value strings. Not available on all target platforms
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void **launchWebApp** (String *webAppId*, `JSONObject` *params*, boolean *relaunchIfRunning*, `LaunchListener` *listener*)

Launch a web application on the TV.

Related capabilities:

- `WebAppLauncher.Launch`
- `WebAppLauncher.Launch.Params` – if launching with params

Parameters:

- `webAppId` – ID of web app assigned by platform vendor
- `params` – Dictionary of key/value strings. Not available on all target platforms
- `relaunchIfRunning` – If supported on target platform, web app will force relaunch if value true
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void **joinWebApp** (*LaunchSession* *webAppLaunchSession*, `LaunchListener` *listener*)

Join an active web app without launching/relaunching. If the app is not running/joinable, the failure block will be called immediately.

Related capabilities:

- `WebAppLauncher.Send`
- `WebAppLauncher.Receive`

Parameters:

- `webAppLaunchSession` – `LaunchSession` for the web app to be joined
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void **joinWebApp** (String *webAppId*, *LaunchListener listener*)

Join an active web app without launching/relaunching. If the app is not running/joinable, the failure block will be called immediately.

Related capabilities:

- `WebAppLauncher.Send`
- `WebAppLauncher.Receive`

Parameters:

- `webAppId` – Unique identifier for the web app to be joined
- `listener` – (optional) `LaunchListener` with methods to be called on success or failure

void **closeWebApp** (*LaunchSession launchSession*, *ResponseListener <Object> listener*)

Closes a web app with the provided `LaunchSession`.

Related capabilities:

- `WebAppLauncher.Close`

Parameters:

- `launchSession` – `LaunchSession` associated with the web app to be closed
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **pinWebApp** (String *webAppId*, *ResponseListener <Object> listener*)

Parameters:

- `webAppId`
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **unPinWebApp** (String *webAppId*, *ResponseListener <Object> listener*)

Parameters:

- `webAppId`
- `listener` – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **isWebAppPinned** (String *webAppId*, *WebAppPinStatusListener listener*)

Parameters:

- `webAppId`
- `listener` – (optional) `WebAppPinStatusListener` with methods to be called on success or failure

ServiceSubscription <WebAppPinStatusListener> **subscribeIsWebAppPinned** (String *webAppId*, *WebAppPinStatusListener listener*)

Parameters:

- webAppId
- listener – (optional) WebAppPinStatusListener with methods to be called on success or failure

ScreenMirroringControl

`com.connectsdk.service.capability.ScreenMirroringControl`

extends CapabilityMethods

The ScreenMirroringControl capability protocol serves to define the methods required for displaying mobile app screen to LG TV.

Properties

`final String Any = "ScreenMirroringControl.Any"`

`final String ScreenMirroring = "ScreenMirroringControl.ScreenMirroring"`

`final String[] Capabilities = { ScreenMirroring }`

Inner Classes

- *ScreenMirroringStartListener*
- *ScreenMirroringStopListener*
- *ScreenMirroringErrorListener*
- *ScreenMirroringError*

Methods

static int getSdkVersion (Context context) Returns the SDK version as an integer. (e.g., 301002)

Parameters:

- context - Application context

static boolean isCompatibleOsVersion () Checks if the OS version can run the screen mirroring function. The screen mirroring function is supported on Android 10 (Q, API Level 29) or higher.

static boolean isRunning (Context context) Checks if the screen mirroring function is running.

Parameters:

- context - Application context

static boolean isSupportScreenMirroring (String deviceId) Checks if the TV supports the screen mirroring function. Currently, only webOS22 TVs are supported.

Parameters:

- deviceId - Device ID value of the TV

void startScreenMirroring (Context context, Intent projectionData, ScreenMirroringStartListener onStartListener)

Starts the screen mirroring. Each step is passed through the *ScreenMirroringStartListener* callback. Before calling this function, user permission for screen capture must be obtained. This data can be passed as an argument.

Parameters:

- context – Application context
- projectionData - Data to use mediaProjection
- onStartListener - (optional) ScreenMirroringStartListener with methods to be called on success or failure

void startScreenMirroring (Context context, Intent projectionData, Class secondScreenClass, ScreenMirroringStartListener onStartListener) Starts screen mirroring in the same way as the API above. There is a secondScreenClass parameter for dual screens.

Parameters:

- context – Application context
- projectionData - Data to use mediaProjection
- secondScreenClass - Screen object to use dual screen
- onStartListener - (optional) ScreenMirroringStartListener with methods to be called on success or failure

void stopScreenMirroring (Context context, ScreenMirroringStopListener stopListener) Stops the screen mirroring. The result is delivered through the *ScreenMirroringStopListener* callback.

Parameters:

- context – Application context
- stopListener - (optional) ScreenMirroringStopListener with methods to be called on success or failure

void setErrorListener (Context context, ScreenMirroringErrorListener errorListener) Designates a *ScreenMirroringErrorListener* to check if an error occurs during execution.

Parameters:

- context – Application context
- errorListener - ScreenMirroringErrorListener to be called when an error occurs

RemoteCameraControl

```
com.connectsdk.service.capability.RemoteCameraControl
extends CapabilityMethods
```

Properties

```
String Any = "RemoteCameraControl.Any"
```

```
String RemoteCamera = "RemoteCameraControl.RemoteCamera"
```

```
String[] Capabilities = { RemoteCamera }
```

```
int LENS_FACING_FRONT = CameraCharacteristics.LENS_FACING_FRONT
```

```
int LENS_FACING_BACK = CameraCharacteristics.LENS_FACING_BACK
```

Inner Classes

- *RemoteCameraStartListener*
- *RemoteCameraStopListener*
- RemoteCameraPlayingListener
- *RemoteCameraPropertyChangeListener*
- *RemoteCameraErrorListener*

Methods

static int getSdkVersion (Context context) Returns the SDK version as an integer. (e.g., 301002)

Parameters:

- context - Application context

static boolean isCompatibleOsVersion () Checks if the OS version can run the remote camera function. The remote camera function is supported on Android 7 (N, API Level 24) or higher.

static boolean isRunning (Context context) Checks if the remote camera function is running.

Parameters:

- context - Application context

static boolean isSupportRemoteCamera (String deviceId) Checks if the TV supports the remote camera function. Currently, only webOS22 TVs are supported.

Parameters:

- deviceId - Device ID value of the TV

void startRemoteCamera (Context context, Surface previewSurface, boolean micMute, int lensFacing, RemoteCameraStartListener startListener) Starts the remote camera. Each step is passed through the *RemoteCameraStartListener* callback.

Parameters:

- context – Application context
- previewSurface - SurfaceView to show a camera preview
- micMute - Microphone mute settings
- lensFacing - Camera lens direction
- startListener - (optional) RemoteCameraStartListener with methods to be called on success or failure

void stopRemoteCamera (Context context, RemoteCameraStopListener stopListener); Stops the remote camera. The result is passed through the *RemoteCameraStopListener* callback.

Parameters:

- context – Application context
- stopListener - (optional) RemoteCameraStopListener with methods to be called on success or failure

void setMicMute (Context context, boolean micMute) Sets the mute function of the microphone.

Parameters:

- context - Application context
- micMute - Microphone mute settings

void setLensFacing (Context *context*, int *lensFacing*) Sets the front/rear camera lens use.

Parameters:

- context - Application context
- lensFacing - Camera lens direction

void setCameraPlayingListener (Context *context*, RemoteCameraPlayingListener *playingListener*) Calls when starting play by selecting a remote camera on the TV.

Parameters:

- context - Application context
- playingListener - RemoteCameraPlayingListener to be called when the camera playback starts on the TV

void setPropertyChangeListener (Context *context*, RemoteCameraPropertyChangeListener *propertyChangeListener*) Calls when camera properties such as brightness and white balance are changed.

Parameters:

- context - Application context
- propertyChangeListener - RemoteCameraPropertyChangeListener to be called when camera properties are changed on the TV

void setErrorListener (Context *context*, ScreenMirroringErrorListener *errorListener*) Calls when an error occurs while running the remote camera.

Parameters:

- context – Application context
- errorListener - RemoteCameraErrorListener to be called when an error occurs

5.10.5 Capability Listeners

AppInfoListener

`com.connectsdk.service.capability.Launcher.AppInfoListener`

extends ResponseListener

Success listener that is called upon requesting info about the current running app.

Passes an AppInfo object containing info about the running app

Inherited Methods

void onSuccess (T *object*) Returns the success of the call of type T.

Parameters:

- object – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError* *error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- error – ServiceCommandError describing the error

AppLaunchListener

`com.connectsdk.service.capability.Launcher.AppLaunchListener`

extends ResponseListener

Success listener that is called upon successfully launching an app.

Passes a LaunchSession Object containing important information about the app's launch session

Inherited Methods

void onSuccess (T *object*) Returns the success of the call of type T.

Parameters:

- *object* – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- *error* – ServiceCommandError describing the error

AppListListener

`com.connectsdk.service.capability.Launcher.AppListListener`

extends ResponseListener

Success block that is called upon successfully getting the app list.

Passes a List containing an AppInfo for each available app on the device

Inherited Methods

void onSuccess (T *object*) Returns the success of the call of type T.

Parameters:

- *object* – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- *error* – ServiceCommandError describing the error

AppStateListener

`com.connectsdk.service.capability.Launcher.AppStateListener`

extends ResponseListener

Success block that is called upon successfully getting an app's state.

Passes an AppState object which contains information about the running app.

Inherited Methods

void onSuccess (T *object*) Returns the success of the call of type T.

Parameters:

- *object* – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- *error* – ServiceCommandError describing the error

ChannelListListener

`com.connectsdk.service.capability.TVControl.ChannelListListener`

extends ResponseListener

Success block that is called upon successfully getting the channel list.

Passes a List of ChannelList objects for each available channel on the TV

Inherited Methods

void onSuccess (T *object*) Returns the success of the call of type T.

Parameters:

- *object* – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- *error* – ServiceCommandError describing the error

ChannelListener

`com.connectsdk.service.capability.TVControl.ChannelListener`

extends ResponseListener

Success block that is called upon successfully getting the current channel's information.

Passes a ChannelInfo object containing information about the current channel

Inherited Methods

void onSuccess (T *object*) Returns the success of the call of type T.

Parameters:

- *object* – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- error – ServiceCommandError describing the error

DurationListener

`com.connectsdk.service.capability.MediaControl.DurationListener`

extends ResponseListener

Success block that is called upon successfully getting the media file's duration.

Passes the duration of the current media file, in seconds

Inherited Methods

void onSuccess (*T object*) Returns the success of the call of type T.

Parameters:

- object – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- error – ServiceCommandError describing the error

ErrorListener

`com.connectsdk.service.capability.listeners.ErrorListener`

Generic asynchronous operation response error handler block. In all cases, you will get a valid ServiceCommandError object. Connect SDK will make all attempts to give you the lowest-level error possible. In cases where an error is generated by Connect SDK, an enumerated error code (ConnectStatusCode) will be present on the ServiceCommandError object.

Low-level error example

Situation

Connect SDK receives invalid XML from a device, generating a parsing error

Result

Connect SDK will call the ErrorListener and pass off the ServiceCommandError generated during parsing of the XML.

High-level error example

Situation

An invalid value is passed to a device capability method

Result

The capability method will immediately invoke the `ErrorListener` and pass off an `ServiceCommandError` object with a status code of `ConnectStatusCodeArgumentError`.

- `error`
`ServiceCommandError` object describing the nature of the problem. Error descriptions are not localized and mostly intended for developer use. It is not recommended to display most error descriptions in UI elements.

Methods

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- `error` – `ServiceCommandError` describing the error

ExternalInputListListener

`com.connectsdk.service.capability.ExternalInputControl.ExternalInputListListener`
extends ResponseListener

Success block that is called upon successfully getting the external input list.

Passes a list containing an `ExternalInputInfo` object for each available external input on the device

Inherited Methods

void onSuccess (T object) Returns the success of the call of type T.

Parameters:

- `object` – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- `error` – `ServiceCommandError` describing the error

MediaInfoListener

`com.connectsdk.service.capability.MediaPlayer.MediaInfoListener`
extends ResponseListener

Inherited Methods

void onSuccess (T object) Returns the success of the call of type T.

Parameters:

- `object` – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- error – ServiceCommandError describing the error

MediaPlayer.LaunchListener

`com.connectsdk.service.capability.MediaPlayer.LaunchListener`

extends ResponseListener

Success block that is called upon successfully playing/displaying a media file.

Passes a MediaLaunchObject which contains the objects for controlling media playback.

Inherited Methods

void onSuccess (*T object*) Returns the success of the call of type T.

Parameters:

- object – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- error – ServiceCommandError describing the error

MuteListener

`com.connectsdk.service.capability.VolumeControl.MuteListener`

extends ResponseListener

Success block that is called upon successfully getting the device's system mute status.

Passes current system mute status

Inherited Methods

void onSuccess (*T object*) Returns the success of the call of type T.

Parameters:

- object – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- error – ServiceCommandError describing the error

PlayStateListener

`com.connectsdk.service.capability.MediaControl.PlayStateListener`

extends ResponseListener

Success block that is called upon any change in a media file's play state.

Passes a PlayStateStatus enum of the current media file

Inherited Methods

void onSuccess (T *object*) Returns the success of the call of type T.

Parameters:

- *object* – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- *error* – ServiceCommandError describing the error

PositionListener

`com.connectsdk.service.capability.MediaControl.PositionListener`

extends ResponseListener

Success block that is called upon successfully getting the media file's current playhead position.

Passes the position of the current playhead position of the current media file, in seconds

Inherited Methods

void onSuccess (T *object*) Returns the success of the call of type T.

Parameters:

- *object* – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- *error* – ServiceCommandError describing the error

ProgramInfoListener

`com.connectsdk.service.capability.TVControl.ProgramInfoListener`

extends ResponseListener

Success block that is called upon successfully getting the current program's information.

Passes a ProgramInfo object containing information about the current program

Inherited Methods

void onSuccess (T *object*) Returns the success of the call of type T.

Parameters:

- *object* – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*) Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- *error* – ServiceCommandError describing the error

ProgramListListener

`com.connectsdk.service.capability.TVControl.ProgramListListener`

extends ResponseListener

Success block that is called upon successfully getting the program list for the current channel.

Passes a ProgramList containing a ProgramInfo object for each available program on the TV's current channel

Inherited Methods

void onSuccess (T *object*)

Returns the success of the call of type T.

Parameters:

- *object* – Response object, can be any number of object types, depending on the protocol/capability/etc

void onError (*ServiceCommandError error*)

Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- *error* – ServiceCommandError describing the error

ResponseListener

`com.connectsdk.service.capability.listeners.ResponseListener`

extends ErrorListener

Generic asynchronous operation response success handler block. If there is any response data to be processed, it will be provided via the responseObject parameter.

- **responseObject** Contains the output data as a generic object reference. This value may be any of a number of types as defined by T in subclasses of ResponseListener. It is also possible that responseObject will be nil for operations that don't require data to be returned (move mouse, send key code, etc).

Methods

void **onSuccess** (T *object*)

Returns the success of the call of type T.

Parameters:

- object – Response object, can be any number of object types, depending on the protocol/capability/etc

Inherited Methods

void **onError** (*ServiceCommandError error*)

Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- error – ServiceCommandError describing the error

State3DModeListener

`com.connectsdk.service.capability.TVControl.State3DModeListener`

extends ResponseListener

Success block that is called upon successfully getting the TV's 3D mode

Passes a Boolean to see Whether 3D mode is currently enabled on the TV

Inherited Methods

void **onSuccess** (T *object*)

Returns the success of the call of type T.

Parameters:

- object – Response object, can be any number of object types, depending on the protocol/capability/etc

void **onError** (*ServiceCommandError error*)

Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- error – ServiceCommandError describing the error

TextInputStatusListener

`com.connectsdk.service.capability.TextInputControl.TextInputStatusListener`

extends ResponseListener

Response block that is fired on any change of keyboard visibility.

Passes `TextInputStatusInfo` object that provides keyboard type & visibility information

Inherited Methods

void **onSuccess** (*T object*)

Returns the success of the call of type T.

Parameters:

- object – Response object, can be any number of object types, depending on the protocol/capability/etc

void **onError** (*ServiceCommandError error*)

Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- error – `ServiceCommandError` describing the error

VolumeListener

`com.connectsdk.service.capability.VolumeControl.VolumeListener`

extends ResponseListener

Success block that is called upon successfully getting the device's system volume.

Passes the current system volume, value is a float between 0.0 and 1.0

Inherited Methods

void **onSuccess** (*T object*)

Returns the success of the call of type T.

Parameters:

- object – Response object, can be any number of object types, depending on the protocol/capability/etc

void **onError** (*ServiceCommandError error*)

Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- error – `ServiceCommandError` describing the error

VolumeStatusListener

`com.connectsdk.service.capability.VolumeControl.VolumeStatusListener`

extends ResponseListener

Success block that is called upon successfully getting the device's system volume status.

Passes current system mute status

Inherited Methods

void **onSuccess** (*T object*)

Returns the success of the call of type T.

Parameters:

- object – Response object, can be any number of object types, depending on the protocol/capability/etc

void **onError** (*ServiceCommandError error*)

Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- error – ServiceCommandError describing the error

ScreenMirroringStartListener

```
com.connectsdk.service.capability.ScreenMirroringControl.  
ScreenMirroringStartListener
```

Methods

void onPairing () Calls for pairing. Pairing is required when connecting to a TV for the first time. When a pairing callback occurs, the app must notify the user by displaying a pop-up with information.

void onStart (boolean result, SecondScreen secondScreen) Calls when screen mirroring starts. The mirroring start result is passed as a result parameter.

Parameters:

- result - Screen mirroring start result
- secondScreen - Virtual second screen for dual screen

ScreenMirroringStopListener

```
com.connectsdk.service.capability.ScreenMirroringControl.  
ScreenMirroringStopListener
```

Method

void onStop (boolean result) Calls when the remote camera is stopped. Returns true if the screen mirroring has been stopped normally and returns false in the following cases.

- If screen mirroring is not running
- Other parameter abnormalities

Parameters:

- result - Screen mirroring stop result

ScreenMirroringErrorListener

`com.connectsdk.service.capability.ScreenMirroringControl.
ScreenMirroringErrorListener`

Method

void onError (ScreenMirroringError *ScreenMirroringError*) Calls when an error occurs during execution. For error types, refer to *ScreenMirroringError*.

Parameters:

- ScreenMirroringError - Screen mirroring error

RemoteCameraStartListener

`com.connectsdk.service.capability.RemoteCameraControl.RemoteCameraStartListener`

Methods

void onPairing () Calls for pairing. Pairing is required when connecting to a TV for the first time. When a pairing callback occurs, the app must notify the user by displaying a pop-up with information.

void onStart (boolean *result*) Calls when the remote camera is connected to the TV. In this state, the remote camera is only connected to the TV, and the camera screen is not displayed.

Parameters:

- result - Remote camera start result

RemoteCameraStopListener

`com.connectsdk.service.capability.RemoteCameraControl.RemoteCameraStopListener`

Method

void onStop (boolean *result*) Calls when the remote camera is stopped. Returns true if the remote camera has been stopped normally and returns false in the following cases.

- If the remote camera is not running
- Other parameter abnormalities

Parameters:

- result - Remote camera stop result

RemoteCameraPropertyChangeListener

`com.connectsdk.service.capability.RemoteCameraControl.RemoteCameraPropertyChangeListener`

Method

void onChange (RemoteCameraProperty *property*) Calls when a camera setting such as brightness or AWB on the TV is changed. For the property types, refer to *RemoteCameraProperty*.

Parameters:

- *property* - Remote camera property

RemoteCameraErrorListener

`com.connectsdk.service.capability.RemoteCameraControl.RemoteCameraErrorListener`

Method

void onError (RemoteCameraError *error*) Calls when an error occurs during execution. For error types, refer to *RemoteCameraError*.

Parameters:

- *error* - Remote camera error

5.10.6 Errors

FireTVServiceError

`com.connectsdk.service.command.FireTVServiceError`

extends ServiceCommandError

This class implements an exception for FireTVService

Methods

FireTVServiceError (String *message*) **Parameters:**

- *message*

FireTVServiceError (String *message*, Throwable *e*) **Parameters:**

- *message*
- *e*

Inherited Methods

ServiceCommandError ()

int **getCode ()**

Object **getPayload ()**

static ServiceCommandError notSupported () Create an error which indicates that feature is not supported by a service

Returns: NotSupportedServiceCommandError

static *ServiceCommandError* getError (int code) Create an error from HTTP response code

Parameters:

- code – HTTP response code

Returns: ServiceCommandError

NotSupportedServiceCommandError

`com.connectsdk.service.command.NotSupportedServiceCommandError`

extends ServiceCommandError

This class defines an Error which is thrown if feature is not supported by a service implementation

Inherited Methods

ServiceCommandError ()

int getCode ()

Object getPayload ()

static *ServiceCommandError* notSupported () Create an error which indicates that feature is not supported by a service

Returns: NotSupportedServiceCommandError

static *ServiceCommandError* getError (int code) Create an error from HTTP response code

Parameters:

- code – HTTP response code

Returns: ServiceCommandError

ServiceCommandError

`com.connectsdk.service.command.ServiceCommandError`

This class implements base service error which is based on HTTP response codes

Methods

ServiceCommandError ()

ServiceCommandError (String detailMessage)

Parameters:

- detailMessage

ServiceCommandError (int code, String detailMessage)

Parameters:

- code
- detailMessage

ServiceCommandError (int code, String desc, Object payload)

Parameters:

- code
- desc
- payload

int **getCode** ()

Object **getPayload** ()

static *ServiceCommandError* **notSupported** ()

Create an error which indicates that feature is not supported by a service

Returns: NotSupportedServiceCommandError

static *ServiceCommandError* **getError** (int *code*)

Create an error from HTTP response code

Parameters:

- code – HTTP response code

Returns: ServiceCommandError

5.10.7 Sessions

LaunchSession

`com.connectsdk.service.sessions.LaunchSession`

Any time anything is launched onto a first screen device, there will be important session information that needs to be tracked. LaunchSession will track this data, and must be retained to perform certain actions within the session.

Inner Classes

- *LaunchSessionType*

Methods

LaunchSession ()

String getAppId () System-specific, unique ID of the app (ex. youtube.leanback.v4, 0000134, hulu)

void setAppId (String *appId*) Sets the system-specific, unique ID of the app (ex. youtube.leanback.v4, 0000134, hulu)

Parameters:

- appId – Id of the app

String getAppName () User-friendly name of the app (ex. YouTube, Browser, Hulu)

void setAppName (String *appName*) Sets the user-friendly name of the app (ex. YouTube, Browser, Hulu)

Parameters:

- appName – Name of the app

String getSessionId () Unique ID for the session (only provided by certain protocols)

void setSessionId (String sessionId) Sets the session id (only provided by certain protocols)

Parameters:

- sessionId – Id of the current session

DeviceService getService () DeviceService responsible for launching the session.

void setService (DeviceService service) DeviceService responsible for launching the session.

Parameters:

- service – Sets the DeviceService

Object getRawData () Raw data from the first screen device about the session. In most cases, this is a JSONObject.

void setRawData (Object rawData) Sets the raw data from the first screen device about the session. In most cases, this is a JSONObject.

Parameters:

- rawData – Sets the raw data

LaunchSessionType getSessionType () When closing a LaunchSession, the DeviceService relies on the sessionType to determine the method of closing the session.

void setSessionType (LaunchSessionType sessionType) Sets the LaunchSessionType of this LaunchSession.

Parameters:

- sessionType – The type of LaunchSession

void close (ResponseListener <Object> listener) Close the app/media associated with the session.

Parameters:

- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

boolean equals (Object launchSession) Compares two LaunchSession objects.

Parameters:

- launchSession – LaunchSession object to compare.

Returns: true if both LaunchSession id and sessionId values are equal

static LaunchSession launchSessionForAppId (String appId) Instantiates a LaunchSession object for a given app ID.

Parameters:

- appId – System-specific, unique ID of the app

Inherited Methods

JSONObject toJSONObject ()

void fromJSONObject (JSONObject obj) **Parameters:**

- obj

LaunchSessionType

`com.connectsdk.service.sessions.LaunchSession.LaunchSessionType`

LaunchSession type is used to help DeviceService's know how to close a LunchSession.

Properties

Unknown Unknown LaunchSession type, may be unable to close this launch session

App LaunchSession represents a launched app

ExternalInputPicker LaunchSession represents an external input picker that was launched

Media LaunchSession represents a media app

WebApp LaunchSession represents a web app

StatusListener

`com.connectsdk.service.sessions.WebAppSession.StatusListener`

extends ResponseListener

Success block that is called upon successfully getting a web app's status.

Passes a WebAppStatus of the current running & foreground status of the web app

Inherited Methods

void **onSuccess** (*T object*)

Returns the success of the call of type T.

Parameters:

- **object** – Response object, can be any number of object types, depending on the protocol/capability/etc

void **onError** (*ServiceCommandError error*)

Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- **error** – ServiceCommandError describing the error

WebAppPinStatusListener

`com.connectsdk.service.sessions.WebAppSession.WebAppPinStatusListener`

extends ResponseListener

Success block that is called upon successfully getting a web app's status.

- **status** The current running & foreground status of the web app

Inherited Methods

void **onSuccess** (*T object*)

Returns the success of the call of type T.

Parameters:

- object – Response object, can be any number of object types, depending on the protocol/capability/etc

void **onError** (*ServiceCommandError error*)

Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- error – ServiceCommandError describing the error

WebAppSession

`com.connectsdk.service.sessions.WebAppSession`

Overview When a web app is launched on a first screen device, there are

certain tasks that can be performed with that web app. WebAppSession serves as a second screen reference of the web app that was launched. It behaves similarly to LaunchSession, but is not nearly as static.

In Depth On top of maintaining session information (contained in the

launchSession property), WebAppSession provides access to a number of capabilities. - MediaPlayer - MediaControl - Bi-directional communication with web app

MediaPlayer and MediaControl are provided to allow for the most common first screen use cases a media player (audio, video, & images).

The Connect SDK JavaScript Bridge has been produced to provide normalized support for these capabilities across protocols (Chromecast, webOS, etc).

Properties

LaunchSession launchSession LaunchSession object containing key session information. Much of this information is required for web app messaging & closing the web app.

Inner Classes

- *LaunchListener* <and-webappsession-launchlistener>
- *StatusListener* <and-statuslistener>
- *WebAppPinStatusListener* <and-webapppinstatuslistener>
- *WebAppStatus* <and-webappstatus>

Methods

WebAppSession (*LaunchSession launchSession, DeviceService service*)

Instantiates a WebAppSession object with all the information necessary to interact with a web app.

Parameters:

- launchSession – LaunchSession containing info about the web app session
- service – DeviceService that was responsible for launching this web app

ServiceSubscription <MessageListener> **subscribeWebAppStatus** (MessageListener listener)

Subscribes to changes in the web app's status.

Parameters:

- listener – (optional) MessageListener to be called on app status change

void **connect** (*ResponseListener* <Object> connectionListener)

Establishes a communication channel with the web app.

Parameters:

- connectionListener – (optional) ResponseListener to be called on success

void **join** (*ResponseListener* <Object> connectionListener)

Establishes a communication channel with a currently running web app.

Parameters:

- connectionListener

void **disconnectFromWebApp** ()

Closes any open communication channel with the web app.

void **pinWebApp** (String webAppId, *ResponseListener* <Object> listener)

Pin the web app on the launcher.

Parameters:

- webAppId
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **unPinWebApp** (String webAppId, *ResponseListener* <Object> listener)

UnPin the web app on the launcher.

Parameters:

- webAppId – NSString webAppId to be unpinned.
- listener – (optional) ResponseListener< Object > with methods to be called on success or failure

void **isWebAppPinned** (String webAppId, *WebAppPinStatusListener* listener)

To check if the web app is pinned or not

Parameters:

- webAppId
- listener – (optional) WebAppPinStatusListener with methods to be called on success or failure

ServiceSubscription <*WebAppPinStatusListener*> **subscribeIsWebAppPinned** (String *webAppId*, *WebAppPinStatusListener* *listener*)

Subscribe to check if the web app is pinned or not

Parameters:

- *webAppId*
- *listener* – (optional) *WebAppPinStatusListener* with methods to be called on success or failure

void **close** (*ResponseListener* <Object> *listener*)

Closes the web app on the first screen device.

Parameters:

- *listener* – (optional) *ResponseListener* to be called on success

void **sendMessage** (String *message*, *ResponseListener* <Object> *listener*)

Sends a simple string to the web app. The Connect SDK JavaScript Bridge will receive this message and hand it off as a string object.

Parameters:

- *message*
- *listener* – (optional) *ResponseListener* to be called on success

void **sendMessage** (JSONObject *message*, *ResponseListener* <Object> *listener*)

Sends a JSON object to the web app. The Connect SDK JavaScript Bridge will receive this message and hand it off as a JavaScript object.

Parameters:

- *message*
- *listener* – (optional) *ResponseListener*< Object > with methods to be called on success or failure

WebAppSessionListener **getWebAppSessionListener** ()

When messages are received from a web app, they are parsed into the appropriate object type (string vs JSON/NSDictionary) and routed to the *WebAppSessionListener*.

void **setWebAppSessionListener** (*WebAppSessionListener* *listener*)

When messages are received from a web app, they are parsed into the appropriate object type (string vs JSON/NSDictionary) and routed to the *WebAppSessionListener*.

Parameters:

- *listener* – *WebAppSessionListener* to be called when messages are received from the web app

Inherited Methods

MediaControl **getMediaControl** ()

Get *MediaControl* implementation

Returns: *MediaControl*

CapabilityPriorityLevel **getMediaControlCapabilityLevel** ()

Get a capability priority for current implementation

Returns: CapabilityPriorityLevel

void **play** (*ResponseListener* <Object> *listener*)

Send play command.

Related capabilities:

- `MediaControl.Play`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **pause** (*ResponseListener* <Object> *listener*)

Send pause command.

Related capabilities:

- `MediaControl.Pause`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **stop** (*ResponseListener* <Object> *listener*)

Send play command.

Related capabilities:

- `MediaControl.Stop`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **rewind** (*ResponseListener* <Object> *listener*)

Send rewind command.

Related capabilities:

- `MediaControl.Rewind`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **fastForward** (*ResponseListener* <Object> *listener*)

Send play command.

Related capabilities:

- `MediaControl.FastForward`

Parameters:

- *listener* – (optional) `ResponseListener< Object >` with methods to be called on success or failure

void **previous** (*ResponseListener* <Object> *listener*)

This method is deprecated. Use `PlaylistControl::previous (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **next** (*ResponseListener* <Object> listener)

This method is deprecated. Use `PlaylistControl::next (ResponseListener<Object> listener)` instead.

Parameters:

- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **seek** (long position, *ResponseListener* <Object> listener)

Seeks to a new position within the current media item

Related capabilities:

- `MediaControl.Seek`

Parameters:

- position – The new position, in milliseconds from the beginning of the stream
- listener – (optional) `ResponseListener<Object>` with methods to be called on success or failure

void **getDuration** (*DurationListener* listener)

Get the current media duration in milliseconds

Parameters:

- listener – (optional) `DurationListener` with methods to be called on success or failure

void **getPosition** (*PositionListener* listener)

Get the current playback position in milliseconds

Parameters:

- listener – (optional) `PositionListener` with methods to be called on success or failure

void **getPlayState** (*PlayStateListener* listener)

Get the current state of playback

Parameters:

- listener – (optional) `PlayStateListener` with methods to be called on success or failure

ServiceSubscription <*PlayStateListener*> **subscribePlayState** (*PlayStateListener* listener)

Subscribe for playback state changes

Parameters:

- listener – receives play state notifications

Returns: `ServiceSubscription<PlayStateListener>`

MediaPlayer **getMediaPlayer** ()

CapabilityPriorityLevel **getMediaPlayerCapabilityLevel** ()

void **getMediaInfo** (*MediaInfoListener* listener)

Parameters:

- listener – (optional) `MediaInfoListener` with methods to be called on success or failure

ServiceSubscription <*MediaInfoListener*> **subscribeMediaInfo** (*MediaInfoListener* listener)

Parameters:

- listener – (optional) `MediaInfoListener` with methods to be called on success or failure

void **displayImage** (*MediaInfo* `mediaInfo`, *LaunchListener* `listener`)

Display an image on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- `MediaPlayer.Display.Image`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- `mediaInfo` – Object of `MediaInfo` class which includes all the information about an image to display.
- listener – (optional) `LaunchListener` with methods to be called on success or failure

void **playMedia** (*MediaInfo* `mediaInfo`, boolean *shouldLoop*, *LaunchListener* `listener`)

Play an audio or video file on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- `MediaPlayer.Play.Video`
- `MediaPlayer.Play.Audio`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- `mediaInfo` – Object of `MediaInfo` class which includes all the information about an image to display.
- `shouldLoop` – Whether to automatically loop playback
- listener – (optional) `LaunchListener` with methods to be called on success or failure

void **closeMedia** (*LaunchSession* `launchSession`, *ResponseListener* `<Object> listener`)

Close a running media session. Because media is handled differently on different platforms, it is required to keep track of `LaunchSession` and `MediaControl` objects to control that media session in the future. `LaunchSession` will be required to close the media and `mediaControl` will be required to control the media.

Related capabilities:

- `MediaPlayer.Close`

Parameters:

- `launchSession` – `LaunchSession` object for use in closing media instance
- `listener` – (optional) `ResponseListener<Object>` with methods to be called on success or failure

PlaylistControl **getPlaylistControl** ()

CapabilityPriorityLevel **getPlaylistControlCapabilityLevel** ()

void **jumpToTrack** (long *index*, *ResponseListener* <Object> *listener*)

Jump the playlist to the designated track.

Play a track specified by index in the playlist

Related capabilities:

- `PlaylistControl.JumpToTrack`

Parameters:

- `index` – index in the playlist, it starts from zero like index of array
- `listener` – optional response listener

void **setPlayMode** (*PlayMode* *playMode*, *ResponseListener* <Object> *listener*)

Set order of playing tracks

Parameters:

- `playMode`
- `listener` – optional response listener

WebAppSession.LaunchListener

`com.connectsdk.service.sessions.WebAppSession.LaunchListener`

extendsResponseListener

Success block that is called upon successfully launch of a web app.

Passes a `WebAppSession` Object containing important information about the web app's session. This object is required to perform many functions with the web app, including app-to-app communication, media playback, closing, etc.

Inherited Methods

void **onSuccess** (*T object*)

Returns the success of the call of type T.

Parameters:

- `object` – Response object, can be any number of object types, depending on the protocol/capability/etc

void **onError** (*ServiceCommandError error*)

Method to return the error that was generated. Will pass an error object with a helpful status code and error message.

Parameters:

- `error` – `ServiceCommandError` describing the error

WebAppSessionListener

`com.connectsdk.service.sessions.WebAppSessionListener`

Methods

void **onReceiveMessage** (*WebAppSession* webAppSession, Object message)

This method is called when a message is received from a web app.

Parameters:

- webAppSession – WebAppSession that corresponds to the web app that sent the message
- message – Object from the web app, either an String or a JSONObject

void **onWebAppSessionDisconnect** (*WebAppSession* webAppSession)

This method is called when a web app's communication channel (WebSocket, etc) has become disconnected.

Parameters:

- webAppSession – WebAppSession that became disconnected

WebAppStatus

`com.connectsdk.service.sessions.WebAppSession.WebAppStatus`

Status of the web app

Properties

Unknown Web app status is unknown

Open Web app is running and in the foreground

Background Web app is running and in the background

Foreground Web app is in the foreground but has not started running yet

Closed Web app is not running and is not in the foreground or background

5.10.8 Info Objects

AppInfo

`com.connectsdk.core.AppInfo`

Normalized reference object for information about a DeviceService's app. This object will, in most cases, be used to launch apps.

In some cases, all that is needed to launch an app is the app id.

Methods

AppInfo () Default constructor method.

AppInfo (String *id*) Default constructor method.

Parameters:

- *id* – App id to launch

String getId () Gets the ID of the app on the first screen device. Format is different depending on the platform. (ex. youtube.leanback.v4, 0000001134, netflix, etc).

void setId (String *id*) Sets the ID of the app on the first screen device. Format is different depending on the platform. (ex. youtube.leanback.v4, 0000001134, netflix, etc).

Parameters:

- *id*

String getName () Gets the user-friendly name of the app (ex. YouTube, Browser, Netflix, etc).

void setName (String *name*) Sets the user-friendly name of the app (ex. YouTube, Browser, Netflix, etc).

Parameters:

- *name*

JSONObject getRawData () Gets the raw data from the first screen device about the app.

void setRawData (JSONObject *data*) Sets the raw data from the first screen device about the app.

Parameters:

- *data*

boolean equals (Object *o*) Compares two AppInfo objects.

Parameters:

- *o* – Other AppInfo object to compare.

Returns: true if both AppInfo id values are equal

Inherited Methods

JSONObject **toJSONObject ()**

AppState

`com.connectsdk.service.capability.Launcher.AppState`

Helper class used with the AppStateListener to return the current state of an app.

Properties

boolean running Whether the app is currently running.

boolean visible Whether the app is currently visible.

Methods

AppState (boolean *running*, boolean *visible*) **Parameters:**

- running
- visible

ChannelInfo

`com.connectsdk.core.ChannelInfo`

Normalized reference object for information about a TV's channels. This object is required to set the channel on a TV.

Methods

ChannelInfo () Default constructor method.

JSONObject getRawData () Gets the raw data from the first screen device about the channel. In most cases, this is an NSDictionary.

void setRawData (JSONObject rawData) Sets the raw data from the first screen device about the channel. In most cases, this is an NSDictionary.

Parameters:

- rawData

String getName () Gets the user-friendly name of the channel

void setName (String channelName) Sets the user-friendly name of the channel

Parameters:

- channelName

String getId () Gets the TV's unique ID for the channel

void setId (String channelId) Sets the TV's unique ID for the channel

Parameters:

- channelId

String getNumber () Gets the TV channel's number (likely to be a combination of the major & minor numbers)

void setNumber (String channelNumber) Sets the TV channel's number (likely to be a combination of the major & minor numbers)

Parameters:

- channelNumber

int getMinorNumber () Gets the TV channel's minor number

void setMinorNumber (int minorNumber) Sets the TV channel's minor number

Parameters:

- minorNumber

int getMajorNumber () Gets the TV channel's major number

void setMajorNumber (int *majorNumber*) Sets the TV channel's major number

Parameters:

- majorNumber

boolean equals (Object *o*) Compares two ChannelInfo objects.

Parameters:

- o

Returns: YES if both ChannelInfo number & name values are equal

Inherited Methods

JSONObject toJSONObject ()

ExternalInputInfo

com.connectsdk.core.ExternalInputInfo

Normalized reference object for information about a DeviceService's external inputs. This object is required to set a DeviceService's external input.

Methods

ExternalInputInfo () Default constructor method.

String getId () Gets the ID of the external input on the first screen device.

void setId (String *inputId*) Sets the ID of the external input on the first screen device.

Parameters:

- inputId

String getName () Gets the user-friendly name of the external input (ex. AV, HDMI1, etc).

void setName (String *inputName*) Sets the user-friendly name of the external input (ex. AV, HDMI1, etc).

Parameters:

- inputName

void setRawData (JSONObject *rawData*) Sets the raw data from the first screen device about the external input.

Parameters:

- rawData

JSONObject getRawData () Gets the raw data from the first screen device about the external input.

boolean isConnected () Whether the DeviceService is currently connected to this external input.

void setConnected (boolean *connected*) Sets whether the DeviceService is currently connected to this external input.

Parameters:

- connected

String getIconURL () Gets the URL to an icon representing this external input.

void setIconURL (String *iconURL*) Sets the URL to an icon representing this external input.

Parameters:

- *iconURL*

boolean equals (Object *o*) Compares two ExternalInputInfo objects.

Parameters:

- *o*

Returns: YES if both ExternalInputInfo id & name values are equal

Inherited Methods

JSONObject toJSONObject ()

ImageInfo

`com.connectsdk.core.ImageInfo`

Normalized reference object for information about an image file. This object can be used to represent a media file (ex. icon, poster)

Inner Classes

- ImageType

Methods

ImageInfo (String *url*) Default constructor method.

Parameters:

- *url*

ImageInfo (String *url*, ImageType *type*, int *width*, int *height*) Default constructor method.

Parameters:

- *url* – add type of file, width and height of image.
- *type*
- *width*
- *height*

String getUrl () Gets URL address of an image file.

void setUrl (String *url*) Sets URL address of an image file.

Parameters:

- *url*

ImageType getType () Gets a type of an image file.

void setType (ImageType type) Sets a type of an image file.

Parameters:

- type

int getWidth () Gets a width of an image.

void setWidth (int width) Sets a width of an image.

Parameters:

- width

int getHeight () Gets a height of an image.

void setHeight (int height) Sets a height of an image.

Parameters:

- height

boolean equals (Object o) **Parameters:**

- o

int hashCode ()

KeyCode

`com.connectsdk.service.capability.KeyControl.KeyCode`

Properties

NUM_0 = (0)

NUM_1 = (1)

NUM_2 = (2)

NUM_3 = (3)

NUM_4 = (4)

NUM_5 = (5)

NUM_6 = (6)

NUM_7 = (7)

NUM_8 = (8)

NUM_9 = (9)

DASH = (10)

ENTER = (11)

Methods

KeyCode (int code) **Parameters:**

- code

int **getCode** ()

static *KeyCode* **createFromInteger** (int *keyCode*) **Parameters:**

- keyCode

MediaInfo

com.connectsdk.core.MediaInfo

Normalized reference object for information about a media to display. This object can be used to pass as a parameter to `displayImage` or `playMedia`.

Inner Classes

- Builder

Methods

MediaInfo (String *url*, String *mimeType*, String *title*, String *description*) This constructor is deprecated. Use `MediaInfo.Builder` instead.

Parameters:

- url – media file
- mimeType – media mime type
- title – optional metadata
- description – optional metadata

MediaInfo (String *url*, String *mimeType*, String *title*, String *description*, List<ImageInfo> *allImages*) This constructor is deprecated. Use `MediaInfo.Builder` instead.

Parameters:

- url – media file
- mimeType – media mime type
- title – optional metadata
- description – optional metadata
- allImages – list of imageInfo objects where [0] is icon, [1] is poster

String getMimeType () Gets type of a media file.

void setMimeType (String *mimeType*) Sets type of a media file.

This method is deprecated.

Parameters:

- mimeType

String getTitle () Gets title for a media file.

void setTitle (String *title*) Sets title of a media file.

This method is deprecated

Parameters:

- title

String getDescription () Gets description for a media.

void setDescription (String *description*) Sets description for a media. This method is deprecated

Parameters:

- description

List<ImageInfo> getImages () Gets list of ImageInfo objects for images representing a media (ex. icon, poster). Where first ([0]) is icon image, and second ([1]) is poster image.

void setImages (List<ImageInfo> *images*) Sets list of ImageInfo objects for images representing a media (ex. icon, poster). Where first ([0]) is icon image, and second ([1]) is poster image.

This method is deprecated

Parameters:

- images

long getDuration () Gets duration of a media file.

void setDuration (long *duration*) Sets duration of a media file. This method is deprecated

Parameters:

- duration

String getUrl () Gets URL address of a media file.

void setUrl (String *url*) Sets URL address of a media file. This method is deprecated

Parameters:

- url

SubtitleInfo **getSubtitleInfo ()**

void addImages (ImageInfo... *images*) Stores ImageInfo objects.

This method is deprecated

Parameters:

- images

MediaLaunchObject

`com.connectsdk.service.capability.MediaPlayer.MediaLaunchObject`

Helper class used with the MediaPlayer.LaunchListener to return the current media playback.

Properties

LaunchSession launchSession The LaunchSession object for the media launched.

MediaControl mediaControl The MediaControl object for the media launched.

***PlaylistControl* playlistControl** The PlaylistControl object for the media launched.

Methods

MediaLaunchObject (*LaunchSession launchSession, MediaControl mediaControl*) **Parameters:**

- launchSession
- mediaControl

MediaLaunchObject (*LaunchSession launchSession, MediaControl mediaControl, PlaylistControl playlistControl*) **Parameters:**

- launchSession
- mediaControl
- playlistControl

PlayMode

`com.connectsdk.service.capability.PlaylistControl.PlayMode`

Enumerates available playlist mode

Properties

Normal Default mode, play tracks in sequence and stop at the end.

Shuffle Shuffle the playlist and play in sequence.

RepeatOne Repeat current track

RepeatAll Repeat entire playlist

PlayStateStatus

`com.connectsdk.service.capability.MediaControl.PlayStateStatus`

Enumerates possible playback status

Properties

Unknown Unknown state

Idle Media source is not set.

Playing Media is playing.

Paused Media is paused.

Buffering Media is buffering on the first screen device (e.g. on the TV)

Methods

static *PlayStateStatus* convertPlayerStateToPlayStateStatus (int *playerState*) Converts int value into PlayStateStatus

Parameters:

- *playerState* – int value

Returns: PlayStateStatus

static *PlayStateStatus* convertTransportStateToPlayStateStatus (String *transportState*) Converts String value into PlayStateStatus

Parameters:

- *transportState* – String value

Returns: PlayStateStatus

ProgramInfo

`com.connectsdk.core.ProgramInfo`

Normalized reference object for information about a TV's program.

Methods

String getId () Gets the ID of the program on the first screen device. Format is different depending on the platform.

void setId (String *id*) Sets the ID of the program on the first screen device. Format is different depending on the platform.

Parameters:

- *id*

String getName () Gets the user-friendly name of the program (ex. Sesame Street, Cosmos, Game of Thrones, etc).

void setName (String *name*) Sets the user-friendly name of the program (ex. Sesame Street, Cosmos, Game of Thrones, etc).

Parameters:

- *name*

***ChannelInfo* getChannelInfo ()** Gets the reference to the ChannelInfo object that this program is associated with

void setChannelInfo (*ChannelInfo* *channelInfo*) Sets the reference to the ChannelInfo object that this program is associated with

Parameters:

- *channelInfo*

Object getRawData () Gets the raw data from the first screen device about the program. In most cases, this is an NSDictionary.

void setRawData (Object *rawData*) Sets the raw data from the first screen device about the program. In most cases, this is an NSDictionary.

Parameters:

- rawData

boolean equals (Object o) Compares two ProgramInfo objects.

Parameters:

- o

Returns: true if both ProgramInfo id & name values are equal

ProgramList

com.connectsdk.core.ProgramList

methods

ProgramList (*ChannelInfo* channel, JSONArray programList)

Parameters

- channel
- programList

ChannelInfo **getChannel()**

JSONArray **getProgramList ()**

JSONObject **toJSONObject ()**

Inherited Methods

JSONObject **toJSONObject ()**

TextInputStatusInfo

com.connectsdk.core.TextInputStatusInfo

Normalized reference object for information about a text input event.

Methods

TextInputStatusInfo ()

boolean **isFocused ()**

void **setFocused** (boolean *focused*)

Parameters:

- focused

TextInputType **getTextInputType ()**

Gets the type of keyboard that should be displayed to the user.

void **setTextInputType** (TextInputType *textInputType*)

Sets the type of keyboard that should be displayed to the user.

Parameters:

- `textInputType`

void **setContentType** (String *contentType*)

Parameters:

- `contentType`

boolean **isPredictionEnabled** ()

void **setPredictionEnabled** (boolean *predictionEnabled*)

Parameters:

- `predictionEnabled`

boolean **isCorrectionEnabled** ()

void **setCorrectionEnabled** (boolean *correctionEnabled*)

Parameters:

- `correctionEnabled`

boolean **isAutoCapitalization** ()

void **setAutoCapitalization** (boolean *autoCapitalization*)

Parameters:

- `autoCapitalization`

boolean **isHiddenText** ()

void **setHiddenText** (boolean *hiddenText*)

Parameters:

- `hiddenText`

JSONObject **getRawData** ()

Gets the raw data from the first screen device about the text input status.

void **setRawData** (JSONObject *data*)

Sets the raw data from the first screen device about the text input status.

Parameters:

- `data`

boolean **isFocusChanged** ()

void **setFocusChanged** (boolean *focusChanged*)

Parameters:

- `focusChanged`

VolumeStatus

`com.connectsdk.service.capability.VolumeControl.VolumeStatus`

Helper class used with the `VolumeControl.VolumeStatusListener` to return the current volume status.

Properties

boolean `isMute`

float `volume`

Methods

VolumeStatus (boolean *isMute*, float *volume*)

Parameters:

- `isMute`
- `volume`

ScreenMirroringError

`com.connectsdk.service.capability.ScreenMirroringControl.ScreenMirroringError`

Enumerates error type

Properties

ERROR_GENERIC The general error

ERROR_CONNECTION_CLOSED The error that occurs when the network is disconnected

ERROR_DEVICE_SHUTDOWN The error that occurs when the TV shuts down

ERROR_RENDERER_TERMINATED The error that occurs when the TV app is closed

ERROR_STOPPED_BY_NOTIFICATION The error that occurs when mirroring is stopped through a notification from the mobile device

RemoteCameraError

`com.connectsdk.service.capability.RemoteCameraControl.RemoteCameraError`

Enumerates error type

Properties

ERROR_GENERIC The general error

ERROR_CONNECTION_CLOSED The error that occurs when the network is disconnected

ERROR_DEVICE_SHUTDOWN The error that occurs when the TV shuts down

ERROR_RENDERER_TERMINATED The error that occurs when the TV app is closed

ERROR_STOPPED_BY_NOTIFICATION The error that occurs when remote camera is stopped through notification from the mobile device

RemoteCameraProperty

`com.connectsdk.service.capability.RemoteCameraControl.RemoteCameraProperty`

Enumerates property type

Properties

UNKNOWN Unknown property

RESOLUTION Property for setting resolution

LENS_FACING Property for setting the front/rear direction of the lens

BRIGHTNESS Property for setting brightness

WHITE_BALANCE Property for setting the white balance

AUTO_WHITE_BALANCE Property for automatic setting of white balance

AUDIO Property for setting audio

5.10.9 Advanced

ConnectableDeviceStore

`com.connectsdk.device.ConnectableDeviceStore`

ConnectableDeviceStore is a interface which can be implemented to save key information about ConnectableDevices that have been connected to. Any class which implements this interface can be used as DiscoveryManager's deviceStore.

A default implementation, DefaultConnectableDeviceStore, will be used by DiscoveryManager if no other ConnectableDeviceStore is provided to DiscoveryManager when startDiscovery is called.

Privacy Considerations

If you chose to implement ConnectableDeviceStore, it is important to keep your users' privacy in mind.

- There should be UI elements in your app to
 - completely disable ConnectableDeviceStore
 - purge all data from ConnectableDeviceStore (`removeAll`)
- Your ConnectableDeviceStore implementation should
 - avoid tracking too much data (indefinitely storing all discovered devices)

periodically remove ConnectableDevices from the ConnectableDeviceStore if they haven't been used/connected in X amount of time

Methods

void addDevice (*ConnectableDevice device*) Add a ConnectableDevice to the ConnectableDeviceStore. If the ConnectableDevice is already stored, it's record will be updated.

Parameters:

- device – ConnectableDevice to add to the ConnectableDeviceStore

void removeDevice (*ConnectableDevice device*) Removes a ConnectableDevice's record from the ConnectableDeviceStore.

Parameters:

- device – ConnectableDevice to remove from the ConnectableDeviceStore

void updateDevice (*ConnectableDevice device*) Updates a ConnectableDevice's record in the ConnectableDeviceStore.

Parameters:

- device – ConnectableDevice to update in the ConnectableDeviceStore

JSONObject getStoredDevices () A JSONObject of all ConnectableDevices in the ConnectableDeviceStore. To get a strongly-typed ConnectableDevice object, use the `getDevice(String)` ; method.

ConnectableDevice getDevice (String uuid) Gets a ConnectableDevice object for a provided id. The id may be for the ConnectableDevice object or any of the DeviceServices.

Parameters:

- uuid – Unique ID for a ConnectableDevice or any of its DeviceService objects

Returns: ConnectableDevice object if a matching uuid was found, otherwise will return null

ServiceConfig getServiceConfig (ServiceDescription serviceDescription) Gets a ServiceConfig object for a provided UUID. This is used by DiscoveryManager to retain crucial service information between sessions (pairing code, etc).

Parameters:

- serviceDescription – Description for the service

Returns: ServiceConfig object if matching description was found, otherwise will return null

void removeAll () Clears out the ConnectableDeviceStore, removing all records.

DefaultConnectableDeviceStore

`com.connectsdk.device.DefaultConnectableDeviceStore`

Default implementation of ConnectableDeviceStore. It stores data in a file in application data directory.

Properties

long created Date (in seconds from 1970) that the ConnectableDeviceStore was created.

long updated Date (in seconds from 1970) that the ConnectableDeviceStore was last updated.

int version Current version of the ConnectableDeviceStore, may be necessary for migrations

long maxStoreDuration = TimeUnit.DAYS.toSeconds(3) Max length of time for a ConnectableDevice to remain in the ConnectableDeviceStore without being discovered. Default is 3 days, and modifications to this value will trigger a scan for old devices.

Methods

void addDevice (*ConnectableDevice* device) Add a *ConnectableDevice* to the *ConnectableDeviceStore*. If the *ConnectableDevice* is already stored, it's record will be updated.

Parameters:

- device – *ConnectableDevice* to add to the *ConnectableDeviceStore*

void removeDevice (*ConnectableDevice* device) Removes a *ConnectableDevice*'s record from the *ConnectableDeviceStore*.

Parameters:

- device – *ConnectableDevice* to remove from the *ConnectableDeviceStore*

void updateDevice (*ConnectableDevice* device) Updates a *ConnectableDevice*'s record in the *ConnectableDeviceStore*.

Parameters:

- device – *ConnectableDevice* to update in the *ConnectableDeviceStore*

void removeAll () Clears out the *ConnectableDeviceStore*, removing all records.

JSONObject getStoredDevices () A *JSONObject* of all *ConnectableDevices* in the *ConnectableDeviceStore*. To get a strongly-typed *ConnectableDevice* object, use the `getDevice (String) ;` method.

***ConnectableDevice* getDevice (String uuid)** Gets a *ConnectableDevice* object for a provided id. The id may be for the *ConnectableDevice* object or any of the *DeviceServices*.

Parameters:

- uuid – Unique ID for a *ConnectableDevice* or any of its *DeviceService* objects

Returns: *ConnectableDevice* object if a matching uid was found, otherwise will return null

ServiceConfig getServiceConfig (ServiceDescription serviceDescription) Gets a *ServiceConfig* object for a provided UUID. This is used by *DiscoveryManager* to retain crucial service information between sessions (pairing code, etc).

Parameters:

- serviceDescription – Description for the service

Returns: *ServiceConfig* object if matching description was found, otherwise will return null

Inherited Methods

void addDevice (*ConnectableDevice* device) Add a *ConnectableDevice* to the *ConnectableDeviceStore*. If the *ConnectableDevice* is already stored, it's record will be updated.

Parameters:

- device – *ConnectableDevice* to add to the *ConnectableDeviceStore*

void removeDevice (*ConnectableDevice* device) Removes a *ConnectableDevice*'s record from the *ConnectableDeviceStore*.

Parameters:

- device – *ConnectableDevice* to remove from the *ConnectableDeviceStore*

void updateDevice (*ConnectableDevice device*) Updates a ConnectableDevice’s record in the ConnectableDeviceStore.

Parameters:

- device – ConnectableDevice to update in the ConnectableDeviceStore

JSONObject getStoredDevices () A JSONObject of all ConnectableDevices in the ConnectableDeviceStore. To get a strongly-typed ConnectableDevice object, use the `getDevice(String)` method.

ConnectableDevice getDevice (String *uuid*) Gets a ConnectableDevice object for a provided id. The id may be for the ConnectableDevice object or any of the DeviceServices.

Parameters:

- uuid – Unique ID for a ConnectableDevice or any of its DeviceService objects

Returns: ConnectableDevice object if a matching uuid was found, otherwise will return null

ServiceConfig getServiceConfig (ServiceDescription *serviceDescription*) Gets a ServiceConfig object for a provided UUID. This is used by DiscoveryManager to retain crucial service information between sessions (pairing code, etc).

Parameters:

- serviceDescription – Description for the service

Returns: ServiceConfig object if matching description was found, otherwise will return null

void removeAll () Clears out the ConnectableDeviceStore, removing all records.

5.11 Getting Started

5.11.1 Setup Instructions

Requirements

This guide assumes basic familiarity with Cordova (PhoneGap), Xcode, and Eclipse. For a more detailed walkthrough of setting up a Cordova project, see the [Cordova platform guides](#).

You should also have:

- Cordova 5.0 or later. We strongly encourage you to use the latest Cordova tools (5.2.0 at the time of this release)
- iOS: Xcode and Xcode command line tools
- Android: Android SDK with “android” tool in PATH or ANDROID_HOME environment variable ([Cordova’s Setup Guide](#))

Creating a Cordova app

Open a command terminal and cd to the directory where you want to create your Cordova project:

```
cordova create hello_connect com.example.helloconnect HelloConnect
```

This will create a directory named “hello_connect” with a basic Cordova app. Use the following commands to create iOS and Android projects:

```
cd hello_connect
cordova platform add android
cordova platform add ios
```

Note: Due to a bug in the current version of Cordova, do not put any spaces in the app name.

Add the Connect SDK Cordova plugin

This will download and install the Connect SDK plugin:

```
cordova plugin add cordova-plugin-connectsdk
```

The plugin will set up the projects automatically. If you run into any issues with the automatic setup process, please email developer@lge.com or file an issue on Github.

5.11.2 Discover & Connect to Device

This guide assumes you're working with a brand new Cordova app as described in the *Setup Instructions*. It will show you how to add a button that selects a supported smart TV on your local WiFi network and displays a video.

Adding a device picker button

Open `hello_connect/www/index.html` in your preferred editor. Let's add a new button:

```
<div class="app">
  <h1>Apache Cordova</h1>
  <button onclick="app.showDevicePicker()">Select a TV</button>
```

Open `hello_connect/www/js/index.js` in your preferred text editor. Find the "onDeviceReady" method, which is called when Cordova is finished initializing. At the end, add the following line:

```
app.setupDiscovery();
```

Next, add a new method to the app object called `setupDiscovery`:

```
setupDiscovery: function () {
  ConnectSDK.discoveryManager.startDiscovery();
}
```

Now let's add a handler for the button:

```
showDevicePicker: function () {
  ConnectSDK.discoveryManager.pickDevice();
}
```

Let's build and run the modified example. If you are building through Xcode/Android Studio you will need to run the following command to update the projects.

```
cordova prepare
```

Otherwise, you can simply build with the Cordova tools</>

```
cordova build
```

Connecting to a device

If the app launch went well, you should be able to click on the “Select a TV” button to bring up a picker.

Next, we should allow the user to actually do something with the TV.

Open `hello_connect/www/js/index.js` again. We’ll modify `showDevicePicker` to talk to the TV by chaining a *success* callback that will be called when a device is selected. This function will be called with a device object as the first argument, which we can use to send a video URL to the TV.

```
showDevicePicker: function () {
    ConnectSDK.discoveryManager.pickDevice().success(function (device) {
        function sendVideo () {
            device.getMediaPlayer().playMedia("http://media.w3.org/2010/05/sintel/
↔trailer.mp4", "video/mp4");
        }

        if (device.isReady()) { // already connected
            sendVideo();
        } else {
            device.on("ready", sendVideo);
            device.connect();
        }
    })
}
```

Capability Filtering

If your app is making use of certain device capabilities (media playback/controls, web app launching, etc), it is strongly recommended that you create filters with this information for `DiscoveryManager`.

Devices that are discovered & shown in the picker will be guaranteed to have the set of capabilities that you have provided. This will prevent your users from selecting a device that has not yet acquired all of its protocols.

```
var videoFilter = new ConnectSDK.CapabilityFilter([
    ConnectSDK.Capabilities.MediaPlayer.Play.Video,
    ConnectSDK.Capabilities.MediaControl.Any,
    ConnectSDK.Capabilities.VolumeControl.UpDown
]);

var imageFilter = new ConnectSDK.CapabilityFilter([
    ConnectSDK.Capabilities.MediaPlayer.Display.Image
]);

ConnectSDK.discoveryManager.setCapabilityFilters([videoFilter, imageFilter]);

app.setupDiscovery();
```

5.12 Developer Guides

5.12.1 Beam Media

A common use case with Connect SDK will be to beam a simple media file (image, video, audio) to a TV. The following is a quick example of how you can beam an image onto a TV. This example is assuming that you have discovered & connected to a device.

Beam an image file

```
var url = "http://www.connectsdk.com/files/9613/9656/8539/test_image.jpg";
var iconUrl = "http://www.connectsdk.com/files/9613/9656/8539/test_image.jpg";
var mimeType = "image/jpeg";

device.getMediaPlayer().displayImage(url, mimeType, {
  title: "Sintel Character Design",
  description: "Blender Open Movie Project",
}).success(function (launchSession, mediaControl) {
  console.log("Image launch successful");
}).error(function (err) {
  console.log("error: " + err.message);
});
```

Beam an audio/video file

```
var myMediaControl;

var url = "http://www.connectsdk.com/files/8913/9657/0225/test_video.mp4";
var iconUrl = "http://www.connectsdk.com/files/7313/9657/0225/test_video_icon.jpg";
var mimeType = "video/mp4";

device.getMediaPlayer().displayImage(url, mimeType, {
  title: "Sintel Trailer",
  description: "Blender Open Movie Project",
}).success(function (launchSession, mediaControl) {
  console.log("Video launch successful");

  // save a reference to the MediaControl object (if supported)
  myMediaControl = mediaControl && mediaControl.acquire();
}).error(function (err) {
  console.log("error: " + err.message);
});
```

Control media playback

In the previous example, you will notice that the success block was called with a mediaControl object. In order to control the media in the current playback session, you will need to store a reference to this mediaControl object and call control methods on that object.

```
// Pause media
myMediaControl.pause()
```

(continues on next page)

(continued from previous page)

```
// Play media
myMediaControl.play();

// Seek to 10 seconds
myMediaControl.seek(10);

// Close media player
myLaunchSession.close();
```

Beam media to web app

A common use case for web apps is the playback and control of media files. Connect SDK provides capabilities for directly playing/controlling media on a `WebAppSession`, provided that web app has integrated the *Connect SDK JavaScript Bridge*.

Rather than calling `playMedia` on your device's `mediaPlayer`, `webAppSession` provides its own `mediaPlayer`. After media has been beamed into the web app, the control is just like any other media session.

```
myWebAppSession.getMediaPlayer().playMedia(url, mimeType, options).success(function_
↪(launchSession, mediaControl) {
    myLaunchSession = launchSession.acquire();
    myMediaControl = mediaControl && mediaControl.acquire();
}).error(function (err) {
    console.log("play video failure: " + err.message);
});
```

Beam a playlist

```
var url = "your-playlist.m3u";
var mimeType = "application/x-mpegurl";
var options = { title: "Playlist", description: "Playlist Description" };

myWebAppSession.getMediaPlayer().playMedia(url, mimeType, options)
.success(function (launchSession, mediaControl, playlistControl) {
    myLaunchSession = launchSession.acquire();
    myMediaControl = mediaControl && mediaControl.acquire();
    myPlaylistControl = playlistControl && playlistControl.acquire();
}).error(function (err) {
    console.log("play video failure: " + err.message);
});
```

Control a playlist

```
// play previous track
myPlaylistControl.previous();
// play next track
myPlaylistControl.next();
// play a track specified by index (index starts from zero)
myPlaylistControl.jumpToTrack(0);
```

5.12.2 Beam Web Apps

There are several platforms available which support the launching of web apps. A web app is typically run on a temporary basis in a full-screen browser instance.

Web App IDs

Both webOS and Chromecast platforms require a web app ID for API calls to launch & communicate with web apps. This web app ID is translated into your web app's URL on web app launch.

For information on creating a web app ID for webOS, please visit the [registration site](#).

To learn how to register for a Chromecast web app ID, visit 'Google's app ID registration site'.

Launch web app with identifier

Connect SDK currently supports web app launching on webOS and Chromecast devices, which both translate a web app identifier into your web app's URL.

Communicate with web app

Bi-directional communication with your web app is made extremely simple. Data can be sent and received strongly-typed as a string or a keyed set of values (JSON object).

```
var webAppId;

if (device.hasService(ConnectSDK.Services.WebOSTV)) {
  webAppId = "5G7328DE";
} else if (device.hasService(ConnectSDK.Services.Chromecast)) {
  webAppId = "3E5106AB";
} else if (device.hasService(ConnectSDK.Services.AirPlay)) {
  webAppId = "http://www.example.com/";
}

if (!webAppId) {
  return;
}

device.getWebAppLauncher().launchWebApp(webAppId).success(function (session) {
  console.log("web app launch success");
}).error(function (err) {
  console.log("web app launch error: " + err.message);
});
```

```
var mySession = null;
var webAppId;

if (device.hasService(ConnectSDK.Services.WebOSTV)) {
  webAppId = "5G7328DE";
} else if (device.hasService(ConnectSDK.Services.Chromecast)) {
  webAppId = "3E5106AB";
}
```

(continues on next page)

(continued from previous page)

```

if (!webAppId) {
    return;
}

device.getWebAppLauncher().launchWebApp(webAppId).success(function (session) {
    // Keep a reference to the session
    mySession = session.acquire();

    // Open a communication channel to the app
    mySession.connect().success(function () {
        console.log("web app connect success");
    }).error(function (err) {
        console.log("web app connect error: " + err.message);
    });

    // Make sure to release the session when done using it
    mySession.on("disconnect", function () {
        mySession.release();
        mySession = null;
    });
}).error(function (err) {
    console.log("web app launch error: " + err.message);
});

```

After successfully establishing a connection, you can send messages to your web app.

```
mySession.sendText("This is a test message");
```

You can also send a Javascript dictionary object which will be received by the web app as an object.

```

var message = {
    someParameter: "someValue",
    anArray: ["array value 1", "array value 2", "array value 3"],
    anotherObject: {
        anotherParameter: "anotherValue"
    }
};

mySession.sendJSON(message);

```

The “message” event allows you to receive messages from your web app.

```

mySession.on("message", function (message) {
    console.log("Received message from web app:" + JSON.stringify(message));
});

```

5.12.3 Launch App on TV

Many TVs and streaming players include support for launching installed apps. The following is a simplified example of how to launch YouTube on a device.

Launch an app

```
device.getLauncher().launchApp("YouTube").success(function (launchSession) {
  console.log("app launch success");
}).error(function (err) {
  console.log("app launch error: " + err.message);
});
```

Device-specific app identifiers

On each device (webOS TV, Roku, etc) apps are identified by different values. Here is an example of the different identifiers in use for the YouTube app.

- webOS: youtube.leanback.v4 (value may change with future updates)
- Netcast: 0000000000017498 (value may be different on each TV)
- DIAL: YouTube (listed in [DIAL registry](#))
- Roku: 837 (Roku-specific channel number)

Launching an app with device-specific identifiers

The following snippet shows how to detect the platform of your device and launch with the appropriate app identifier.

```
var appId;

if (device.hasService(ConnectSDK.Services.WebOSTV)) {
  appId = "youtube.leanback.v4";
} else if (device.hasService(ConnectSDK.Services.NetcastTV)) {
  appId = "0000000000017498";
} else if (device.hasService(ConnectSDK.Services.Roku)) {
  appId = "837";
} else if (device.hasService(ConnectSDK.Services.DIAL)) {
  appId = "YouTube";
}

if (!appId) {
  return;
}

device.getLauncher().launchApp(appId).success(function (launchSession) {
  console.log("app launch success");
}).error(function (err) {
  console.log("app launch error: " + err.message);
});
```

Launching an app with parameters

In most cases, a device's launcher object will allow you to pass launch parameters to your app. Connect SDK has normalized the parameter input type to a keyed set of values. These values are then parsed into the appropriate format for the protocol (XML, JSON, URL params, etc).

```

var params = {
    "someKey": "someValue"
}

device.getLauncher().launchApp(appId, params).success(function (launchSession) {
    console.log("app launch success");
}).error(function (err) {
    console.log("app launch error: " + err.message);
});

```

Important: Due to the variety of protocols in use, it is strongly recommended that you only use strings for the keys AND values of your parameters.

5.12.4 Discovery Manager

At the heart of Connect SDK is DiscoveryManager, a multi-protocol service discovery engine with a pluggable architecture. Much of your initial experience with Connect SDK will be with the DiscoveryManager class, as it consolidates discovered service information into ConnectableDevice objects.

DiscoveryManager supports discovering services of differing protocols by using DiscoveryProviders. Many services are discoverable over SSDP and are registered to be discovered with the SSDPDiscoveryProvider class.

As services are discovered on the network, the DiscoveryProviders will notify DiscoveryManager. DiscoveryManager is capable of attributing multiple services, if applicable, to a single ConnectableDevice instance. Thus, it is possible to have a mixed-mode ConnectableDevice object that is theoretically capable of more functionality than a single service can provide.

DiscoveryManager keeps a running list of all discovered devices and maintains a filtered list of devices that have satisfied any of your CapabilityFilters. This filtered list is used by the DevicePicker when presenting the user with a list of devices.

Connect SDK device discovery can be started in one line.

```
ConnectSDK.discoveryManager.startDiscovery();
```

Features

Filtering devices by capability

It will be necessary in many cases to filter out devices that don't support a desired feature-set. DiscoveryManager provides the setCapabilityFilters method to provide for this ability.

Here is a simple example that discovers devices that support (video playback AND any media controls AND volume up/down) OR (image display).

```

var videoFilter = new ConnectSDK.CapabilityFilter([
    ConnectSDK.Capabilities.MediaPlayer.Play.Video,
    ConnectSDK.Capabilities.MediaControl.Any,
    ConnectSDK.Capabilities.VolumeControl.UpDown
]);

var imageFilter = new ConnectSDK.CapabilityFilter([
    ConnectSDK.Capabilities.MediaPlayer.Display.Image

```

(continues on next page)

(continued from previous page)

```
    ]);  
  
ConnectSDK.discoveryManager.setCapabilityFilters([videoFilter, imageFilter]);  
app.setupDiscovery();
```

Pairing level

Connect SDK has support for pairing with certain devices. Having pairing disabled may reduce the number of supported capabilities that a ConnectableDevice has. Certain devices, although they may support the features you are filtering for, may not pass your CapabilityFilter if pairing is disabled.

See the *Supported Features* list for information on what devices require pairing for certain capabilities.

For the best user experience, Connect SDK has disabled pairing by default. Pairing can be enabled very easily, but it must be enabled before DiscoveryManager is started for the first time.

```
// Include capabilities that require pairing  
ConnectSDK.discoveryManager.setPairingLevel(ConnectSDK.PairingLevel.ON);  
  
// Exclude capabilities that require pairing (this is the default)  
ConnectSDK.discoveryManager.setPairingLevel(ConnectSDK.PairingLevel.OFF);
```

Automatic stop/resume on app state change

If DiscoveryManager is running while your app enters a background state, it will resume immediately upon returning to a foreground state. This is to prevent battery drain on the user's device.

See also:

- *DiscoveryManager*
- *CapabilityFilter*

5.13 API References

5.13.1 Discovery

CapabilityFilter

CapabilityFilter consists of a list of capabilities which must all be present in order for the filter to match.

For example,

```
new ConnectSDK.CapabilityFilter([ConnectSDK.Capabilities.MediaPlayer.Play.Video,  
                                ConnectSDK.Capabilities.MediaControl.Pause])
```

describes a device that supports showing a video and pausing it.

Methods

new **CapabilityFilter** (*capabilities*)

Create a CapabilityFilter

Parameters:

- capabilities (string[]) – array of capabilities

capabilityFilter. **getCapabilities** ()

Returns: string[] – list of capabilities in filter

DevicePicker

DevicePicker represents a picker UI widget created by calling `DiscoveryManager.pickDevice()`.

Example:

```
var devicePicker = ConnectSDK.discoveryManager.pickDevice()
devicePicker.success(function (device) {
    console.log("picked device " + device.getFriendlyName());
});
```

Methods

devicePicker.close () Close the device picker.

Mixin Methods - SimpleEventEmitter

devicePicker.addListener (*event, callback, [context]*) Add event listener.

Parameters:

- event (string) – name of event
- callback (function) – function to call when event is fired
- context (object) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

devicePicker.removeListener (*event, [callback], [context]*) Remove event listener with the specified callback and context. If callback is null or undefined, all callbacks for this event will be removed.

Parameters:

- event (string) – name of event
- callback (function) [optional] – function originally passed to addListener
- context (object) [optional] – context object originally passed to addListener

Returns: object – reference to the same object to allow chaining

devicePicker.on (*event, callback, [context]*) Alias for addListener.

Parameters:

- event (string) – name of event

- callback (function) – function to call when event is fired
- context (object) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

devicePicker.off (*event*, [*callback*], [*context*]) Alias for removeListener.

Parameters:

- event (string) – event name
- callback (function) [optional] – function originally passed to on
- context (object) [optional] – context object originally passed to on

Returns: object – reference to the same object to allow chaining

Mixin Methods - SuccessCallbacks

devicePicker.success (*callback*, [*context*]) Register a callback for the “success” event. The success callback may be called with zero or more arguments depending on the type of response.

Example:

```
obj.success(function (result) {
  this.report("I got a result: " + result);
}, this);
```

Parameters:

- callback (function) – function to call when event is fired
- context (*) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

devicePicker.error (*callback*, [*context*]) Register a callback for the “error” event. The error callback will be called with a error object as the only argument.

Example:

```
obj.error(function (err) {
  this.reportError("I got an error: " + err);
}, this);
```

Parameters:

- callback (function) – function to call when event is fired
- context (*) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

devicePicker.complete (*callback*, [*context*]) Register a callback for the “complete” event. The complete callback will be called with

Example:

```
obj.complete(function (err, result) {
  if (err) {
    this.report("I got an error: " + err);
  } else {
```

(continues on next page)

(continued from previous page)

```

        console.log("I got a result: " + result);
    }
}, this);

```

Parameters:

- callback (function) – function to call when event is fired
- context (*) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

DiscoveryManager

ConnectSDK.discoveryManager is the main entry point into ConnectSDK. It allows finding devices on the local network and displaying a picker to select and connect to a device. DiscoveryManager should always be accessed through its singleton instance, ConnectSDK.discoveryManager.

DiscoveryManager emits the following events while active:

- startdiscovery
- stopdiscovery
- devicelistchanged
- devicefound (device)
- devicelost (device)
- deviceupdated (device)

Methods

discoveryManager.startDiscovery (*config*) Start searching for devices. DiscoveryManager will start emitting events as the device list changes, and populates the device list used by pickDevice().

Parameters:

- config (Object) [optional] – Dictionary of settings to configure before starting discovery. Supported keys are “pairingLevel” and “capabilityFilters”. See setPairingLevel and setCapabilityFilter for more details.

discoveryManager.stopDiscovery () Stop searching for devices.

discoveryManager.setPairingLevel (*pairingLevel*) Set pairing level. If set to ConnectSDK.PairingLevel.OFF, the SDK will request device capabilities that do not require entering a pairing code/confirmation.

Parameters:

- pairingLevel (string) – Valid values are the constants ConnectSDK.PairingLevel.ON and ConnectSDK.PairingLevel.OFF

discoveryManager.setAirPlayServiceMode () Set mode for AirPlay support. If set to ConnectSDK.AirPlayServiceMode.WebApp, a web app will be mirrored to the TV. If set to ConnectSDK.AirPlayServiceMode.Media, only media APIs will be available. On Android, media mode is the only option.

NOTE: This setting must be configured before calling startDiscovery(), or passed in the options parameter to startDiscovery(). The mode should not be changed once configured.

discoveryManager.setCapabilityFilters (*filters*) Set capability filters. DiscoveryManager will only show devices that match at least one of the CapabilityFilter instances.

Example:

```
// Show devices that support playing videos and pausing OR support launching
↳ YouTube with a video id
ConnectSDK.discoveryManager.setCapabilityFilters([
  new ConnectSDK.CapabilityFilter([ConnectSDK.Capabilities.MediaPlayer.Play.
↳ Video, ConnectSDK.Capabilities.MediaControl.Pause])
  new ConnectSDK.CapabilityFilter([ConnectSDK.Capabilities.Launcher.YouTube.
↳ Params])
])
```

Parameters:

- filters (CapabilityFilter[]) – array of CapabilityFilter objects

discoveryManager.pickDevice (*options*) Show device picker popup. To get notified when the user has selected a device, add a success/error listener to the DevicePicker returned when calling this method.

Parameters:

- options (Object) [optional] – All keys are optional

```
- pairingType (string): PairingType to use
```

Returns: *DevicePicker*

discoveryManager.getDeviceList () Get a list of discovered devices available on the network.

Returns: *ConnectableDevice*[]

Mixin Methods - SimpleEventEmitter

discoveryManager.addListener (*event, callback, [context]*) Add event listener.

Parameters:

- event (string) – name of event
- callback (function) – function to call when event is fired
- context (object) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

discoveryManager.removeListener (*event, [callback], [context]*) Remove event listener with the specified callback and context. If callback is null or undefined, all callbacks for this event will be removed.

Parameters:

- event (string) – name of event
- callback (function) [optional] – function originally passed to addListener
- context (object) [optional] – context object originally passed to addListener

Returns: object – reference to the same object to allow chaining

discoveryManager.on (*event, callback, [context]*) Alias for addListener.

Parameters:

- event (string) – name of event

- callback (function) – function to call when event is fired
- context (object) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

discoveryManager.off (*event*, [*callback*], [*context*]) Alias for removeListener.

Parameters:

- event (string) – event name
- callback (function) [optional] – function originally passed to on
- context (object) [optional] – context object originally passed to on

Returns: object – reference to the same object to allow chaining

5.13.2 Device

Command

Command objects are returned when calling capability methods. Command objects allow listening for success/cancel events from the request.

Example:

```
var command = device.getLauncher().launchBrowser(url);

command.success(function (launchSession) {
    console.log("command was successful");
}).error(function (err) {
    console.error("command failed");
});
```

Mixin Methods - SimpleEventEmitter

command.addListener (*event*, *callback*, [*context*]) Add event listener.

Parameters:

- event (string) – name of event
- callback (function) – function to call when event is fired
- context (object) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

command.removeListener (*event*, [*callback*], [*context*]) Remove event listener with the specified callback and context. If callback is null or undefined, all callbacks for this event will be removed.

Parameters:

- event (string) – name of event
- callback (function) [optional] – function originally passed to addListener
- context (object) [optional] – context object originally passed to addListener

Returns: object – reference to the same object to allow chaining

command.on (*event*, *callback*, [*context*]) Alias for addListener.

Parameters:

- event (string) – name of event
- callback (function) – function to call when event is fired
- context (object) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

command.off (*event*, [*callback*], [*context*]) Alias for removeListener.

Parameters:

- event (string) – event name
- callback (function) [optional] – function originally passed to on
- context (object) [optional] – context object originally passed to on

Returns: object – reference to the same object to allow chaining

Mixin Methods - SuccessCallbacks

command.success (*callback*, [*context*]) Register a callback for the “success” event. The success callback may be called with zero or more arguments depending on the type of response.

Example:

```
obj.success(function (result) {
    this.report("I got a result: " + result);
}, this);
```

Parameters:

- callback (function) – function to call when event is fired
- context (*) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

command.error (*callback*, [*context*]) Register a callback for the “error” event. The error callback will be called with a error object as the only argument.

Example:

```
obj.error(function (err) {
    this.reportError("I got an error: " + err);
}, this);
```

Parameters:

- callback (function) – function to call when event is fired
- context (*) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

command.complete (*callback*, [*context*]) Register a callback for the “complete” event. The complete callback will be called with

Example:

```
obj.complete(function (err, result) {
  if (err) {
    this.report("I got an error: " + err);
  } else {
    console.log("I got a result: " + result);
  }
}, this);
```

Parameters:

- callback (function) – function to call when event is fired
- context (*) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

ConnectableDevice

ConnectableDevice represents a device on the network. It provides several *capability interfaces* which allow the developer to get information from and control the device.

These interfaces are accessed using getter methods like `device.getLauncher()`. Not all of the capabilities or methods are available on every device; you should check if the functionality is supported using `device.supports(capabilityName)`.

If the device was selected from the built-in picker, it will already be connected; if the device was obtained from elsewhere then you must call `device.connect()` and wait for the “ready” event before trying to use the device.

Example:

```
device.on("ready", function () {
  // ready to send commands now
  device.getLauncher().launchYouTube(videoId);
});

device.connect();
```

ConnectableDevice emits the following high-level events:

- ready - device is ready to use
- disconnect - device is no longer connected
- capabilitieschanged - some capabilities may be available or unavailable now

Internally, ConnectableDevice uses one or more *services* to control a device on the network. Services speak a specific protocol like DIAL or DLNA or other vendor-specific protocols. Services are not directly accessible from the Connect SDK Cordova plugin at this time.

There are several events related to the process of connecting to individual services:

- serviceconnectionrequired - pending connection
- serviceconnectionerror - error connecting to a service
- servicepairingrequired - pairing is required for a service
- servicepairingsuccess - pairing successful for a service
- servicepairingerror - error pairing with a service

Methods

connectableDevice.getLauncher () Returns: *Launcher*

connectableDevice.getMediaPlayer () Returns: *MediaPlayer*

connectableDevice.getExternalInputControl () Returns: *ExternalInputControl*

connectableDevice.getMediaControl () Returns: *MediaControl*

connectableDevice.getKeyControl () Returns: *KeyControl*

connectableDevice.getMouseControl () Returns: *MouseControl*

connectableDevice.getTextInputControl () Returns: *TextInputControl*

connectableDevice.getPowerControl () Returns: *PowerControl*

connectableDevice.getToastControl () Returns: *ToastControl*

connectableDevice.getTVControl () Returns: *TVControl*

connectableDevice.getVolumeControl () Returns: *VolumeControl*

connectableDevice.getAppLauncher () Returns: *WebAppLauncher*

connectableDevice.connect () Connect to the device.

connectableDevice.disconnect () Disconnect from the device.

connectableDevice.setPairingType (*pairingType*) Set a desirable pairing type to the device.

Parameters:

- *pairingType* – (string): PairingType to use

connectableDevice.isReady () Returns true if device is ready to use.

connectableDevice.getFriendlyName () Get the human-readable name of the device.

Returns: string

connectableDevice.getIPAddress () Get the last known IP address of the device.

Returns: string

connectableDevice.getModelName () Get the device model name.

Returns: string

connectableDevice.getModelNumber () Get the device model number.

Returns: string

connectableDevice.getCapabilities () Get a list of capabilities supported by this device.

Returns: string[] – array of capabilities supported by this device

connectableDevice.hasCapability (*name*) **Parameters:**

- *name* (string) – of capability. You should use the ConnectSDK.Capabilities constant to reference strings.

Returns: boolean – true if device supports the given capability

connectableDevice.supports ([...]) Flexible version of hasCapability which returns true if all of the capabilities specified are supported.

- supports(ConnectSDK.Capabilities.MediaControl.Any)
- supports(ConnectSDK.Capabilities.VolumeControl.Set, ConnectSDK.Capabilities.Launcher.Any)

- supports([ConnectSDK.Capabilities.TVControl.Any, ConnectSDK.Capabilities.Launcher.Any])

Parameters:

- ... [optional] – array of capability names. You should use the ConnectSDK.Capabilities constant to reference strings.

Returns: boolean – true if all specified capabilities are supported

connectableDevice.supportsAny ([...]) Like supports() but returns true if any specified capability is supported.

Parameters:

- ... [optional] – array of capability names. You should use the ConnectSDK.Capabilities constant to reference strings.

Returns: boolean – true if any specified capability is supported

connectableDevice.hasService (*serviceName*) Returns true if the device supports the specified service. See ConnectSDK.Services for a list of constants.

Parameters:

- serviceName (string)

Returns: boolean – true if service is supported

connectableDevice.getService (*serviceName*) Returns a wrapper for a service which gives access to low-level functionality. Only a limited subset of the services supported by the native SDK are available through this plugin.

Parameters:

- serviceName (string)

Returns: object – service object or null if not supported

connectableDevice.getId () Returns an internal id assigned by the SDK to this device. For devices that have been connected to or paired, this id will be persisted to disk in the device store to allow the app to identify the device later (such as reconnecting to the last connected device when starting the app).

Mixin Methods - SimpleEventEmitter

connectableDevice.addListener (*event, callback, [context]*) Add event listener.

Parameters:

- event (string) – name of event
- callback (function) – function to call when event is fired
- context (object) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

connectableDevice.removeListener (*event, [callback], [context]*) Remove event listener with the specified callback and context. If callback is null or undefined, all callbacks for this event will be removed.

Parameters:

- event (string) – name of event
- callback (function) [optional] – function originally passed to addListener
- context (object) [optional] – context object originally passed to addListener

Returns: object – reference to the same object to allow chaining

connectableDevice.on (*event*, *callback*, [*context*]) Alias for addListener.

Parameters:

- event (string) – name of event
- callback (function) – function to call when event is fired
- context (object) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

connectableDevice.off (*event*, [*callback*], [*context*]) Alias for removeListener.

Parameters:

- event (string) – event name
- callback (function) [optional] – function originally passed to on
- context (object) [optional] – context object originally passed to on

Returns: object – reference to the same object to allow chaining

Subscription

Subscription objects are returned when calling capability subscription methods.

Subscription objects allow listening for success/error events from the request. Success events may be emitted multiple times when updates to the subscription are received.

Example:

```
var subscription = device.getVolumeControl().subscribeVolume();
var updateCount = 0;

subscription.success(function (volume) {
    // this may be called multiple times
    console.log("got volume update: " + volume);

    updateCount++;
    if (updateCount > 5) {
        // unsubscribe after 5 updates
        subscription.unsubscribe();
    }
}).error(function (err) {
    console.error("subscription failed");
});
```

Methods

subscription.unsubscribe () Unsubscribes from this subscription. Notifies the device that updates are no longer needed, and stops emitting events from this Subscription object.

Mixin Methods - SimpleEventEmitter

subscription.addListener (*event*, *callback*, [*context*]) Add event listener.

Parameters:

- event (string) – name of event
- callback (function) – function to call when event is fired
- context (object) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

subscription.removeListener (*event*, [*callback*], [*context*]) Remove event listener with the specified callback and context. If callback is null or undefined, all callbacks for this event will be removed.

Parameters:

- event (string) – name of event
- callback (function) [optional] – function originally passed to addListener
- context (object) [optional] – context object originally passed to addListener

Returns: object – reference to the same object to allow chaining

subscription.on (*event*, *callback*, [*context*]) Alias for addListener.

Parameters:

- event (string) – name of event
- callback (function) – function to call when event is fired
- context (object) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

subscription.off (*event*, [*callback*], [*context*]) Alias for removeListener.

Parameters:

- event (string) – event name
- callback (function) [optional] – function originally passed to on
- context (object) [optional] – context object originally passed to on

Returns: object – reference to the same object to allow chaining

Mixin Methods - SuccessCallbacks

subscription.success (*callback*, [*context*]) Register a callback for the “success” event. The success callback may be called with zero or more arguments depending on the type of response.

Example:

```
obj.success(function (result) {
  this.report("I got a result: " + result);
}, this);
```

Parameters:

- callback (function) – function to call when event is fired
- context (*) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

subscription.error (*callback*, [*context*]) Register a callback for the “error” event. The error callback will be called with a error object as the only argument.

Example:

```
obj.error(function (err) {
  this.reportError("I got an error: " + err);
}, this);
```

Parameters:

- callback (function) – function to call when event is fired
- context (*) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

subscription.complete (*callback*, [*context*]) Register a callback for the “complete” event. The complete callback will be called with

Example:

```
obj.complete(function (err, result) {
  if (err) {
    this.report("I got an error: " + err);
  } else {
    console.log("I got a result: " + result);
  }
}, this);
```

Parameters:

- callback (function) – function to call when event is fired
- context (*) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

5.13.3 Sessions

LaunchSession

A LaunchSession represents the result of an app launch. Its primary purpose is to be able to close an app that was previously launched, using the launchSession.close() method.

Methods

launchSession.close () Close the app/media associated with this launch session.

Mixin Methods - WrappedObject

launchSession.acquire () Indicate that you would like to keep an active reference to this object. Wrapped objects that are not acquired may be freed after the success callback returns.

Returns: object – reference to object

launchSession.release () Release the reference to this object. After calling .release(), this object may no longer be used. You should always release objects when you no longer need them, to avoid memory leaks.

WebAppSession

A WebAppSession represents a web-based app running on a TV. You can communicate with a web app by first calling `connect()` to establish a communication channel, and then listening for “message” events as well as sending your own messages using `sendText` and `sendJSON`.

Example:

```
device.getWebAppLauncher().launchWebApp(webAppId).success(function (session) {
    this.session = session.acquire(); // hold on to a reference

    session.connect().success(function () {
        session.sendText("Hello world");
    });

    session.on('message', function (message) {
        // message could be either a string or an object
        if (typeof message === 'string') {
            console.log("received string message: " + message);
        } else {
            console.log("received object message: " + JSON.stringify(message));
        }
    }, this);

    session.on('disconnect', function () {
        console.log("session disconnected");
        this.session = null;
    }, this);
}, this);
```

Methods

webAppSession.connect () Open a message channel to the app.

Returns: *Command*

webAppSession.disconnect () Close channel to app.

Returns: *Command*

webAppSession.setWebAppSessionListener () Set web app session listener to app

Returns: *Command*

webAppSession.sendText (text) Send a text string to the app. Must be connected first.

Parameters:

- text (string) – Text to send to the app

Returns: *Command*

webAppSession.sendJSON (object) Send a plain JavaScript object to the app. Must be connected first. If the receiving app does not support non-string messages, the object will be serialized into a string in JSON format.

Parameters:

- object (object) – Plain JavaScript object to send to the app

Returns: *Command*

webAppSession.close () Close the web app.

Returns: *Command*

Mixin Methods - SimpleEventEmitter

webAppSession.addListener (*event*, *callback*, [*context*]) Add event listener.

Parameters:

- *event* (string) – name of event
- *callback* (function) – function to call when event is fired
- *context* (object) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

webAppSession.removeListener (*event*, [*callback*], [*context*]) Remove event listener with the specified callback and context. If callback is null or undefined, all callbacks for this event will be removed.

Parameters:

- *event* (string) – name of event
- *callback* (function) [optional] – function originally passed to addListener
- *context* (object) [optional] – context object originally passed to addListener

Returns: object – reference to the same object to allow chaining

webAppSession.on (*event*, *callback*, [*context*]) Alias for addListener.

Parameters:

- *event* (string) – name of event
- *callback* (function) – function to call when event is fired
- *context* (object) [optional] – object to bind to “this” value when calling function

Returns: object – reference to the same object to allow chaining

webAppSession.off (*event*, [*callback*], [*context*]) Alias for removeListener.

Parameters:

- *event* (string) – event name
- *callback* (function) [optional] – function originally passed to on
- *context* (object) [optional] – context object originally passed to on

Returns: object – reference to the same object to allow chaining

Mixin Methods - WrappedObject

webAppSession.acquire () Indicate that you would like to keep an active reference to this object. Wrapped objects that are not acquired may be freed after the success callback returns.

Returns: object – reference to object

webAppSession.release () Release the reference to this object. After calling `.release()`, this object may no longer be used. You should always release objects when you no longer need them, to avoid memory leaks.

5.13.4 Capabilities

ExternalInputControl

The ExternalInputControl capability serves to define the methods required for normalizing all functions regarding external input switching and general info.

ExternalInputInfo objects are plain JavaScript objects with the following properties:

- id (string): A platform-specific id representing an input device
- name (string): A human-readable name for the input device

Methods

externalInputControl.getExternalInputList () Get a list of input devices (HDMI, AV, etc) connected to the device

On success, the success event/callback will be fired with the arguments (externalInputList)

- externalInputList: ExternalInputInfo[]

Related capabilities:

- ExternalInputControl.List

Returns: *Command*

externalInputControl.setExternalInput (externalInputInfo) Switch to the specified external input

Related capabilities:

- ExternalInputControl.Set

Parameters:

- externalInputInfo (object) – Object containing the proper info to set current input. For best cross-platform support, it is suggested to get ExternalInputInfo references from getExternalInputList, if possible.

Returns: *Command*

externalInputControl.showExternalInputPicker () **Returns:** *Command*

KeyControl

The KeyControl capability serves to define the methods required for normalizing common key commands (up, down, left right, ok, back, home, key code).

Methods

keyControl.up () Sends the up button key code to the TV.

Related capabilities:

- KeyControl.Up

Returns: *Command*

keyControl.down () Sends the down button key code to the TV.

Related capabilities:

- KeyControl.Down

Returns: *Command*

keyControl.left () Sends the left button key code to the TV.

Related capabilities:

- `KeyControl.Left`

Returns: *Command*

keyControl.right () Sends the right button key code to the TV.

Related capabilities:

- `KeyControl.Right`

Returns: *Command*

keyControl.ok () Sends the OK button key code to the TV.

Related capabilities:

- `KeyControl.OK`

Returns: *Command*

keyControl.back () Sends the back button key code to the TV.

Related capabilities:

- `KeyControl.Back`

Returns: *Command*

keyControl.home () Sends the home button key code to the TV.

Related capabilities:

- `KeyControl.Home`

Returns: *Command*

keyControl.sendKeyCode (*keyCode*) Sends a key code value to the TV.

Related capabilities:

- `KeyControl.Send.KeyCode`

Parameters:

- `keyCode` (number) – Refer to the native Connect SDK device services for a list of keycodes

Returns: *Command*

Launcher

The Launcher capability protocol serves to define the methods required for normalizing the launching of apps. It allows for in-built support for certain common launch types (deep-linking to YouTube, Netflix, Hulu, browser, etc) as well as by (platform-specific) app id.

Methods

launcher.launchApp (*appId*) Launch an application on the device.

On success, the success event/callback will be fired with the arguments (launchSession)

- `launchSession`: `LaunchSession`

Related capabilities:

- `Launcher.App`

Parameters:

- `appId` (string) – ID of the application

Returns: *Command***launcher.closeApp** (*appId*) Close an application on the device.**Related capabilities:**

- `Launcher.App.Close`

Parameters:

- `appId` (string)

Returns: *Command***launcher.launchAppStore** (*appId*) Launch the device's app store app, optionally deep-linked to a specific app's page.On success, the success event/callback will be fired with the arguments (`launchSession`)

- `launchSession`: `LaunchSession`

Related capabilities:

- `Launcher.AppStore`
- `Launcher.AppStore.Params`

Parameters:

- `appId` (string) – (optional) ID of the application to show in the app store

Returns: *Command***launcher.launchBrowser** (*url*) Launch the web browser. Will launch deep-linked to provided URL, if supported on the target platform.On success, the success event/callback will be fired with the arguments (`launchSession`)

- `launchSession`: `LaunchSession`

Related capabilities:

- `Launcher.Browser`
- `Launcher.Browser.Params` – if launching with url

Parameters:

- `url` (string)

Returns: *Command***launcher.launchHulu** (*contentId*) Launch Hulu app. Will launch deep-linked to provided contentId, if supported on the target platform.On success, the success event/callback will be fired with the arguments (`launchSession`)

- `launchSession`: `LaunchSession`

Related capabilities:

- `Launcher.Hulu`

- `Launcher.Hulu.Params` – if launching with `contentId`

Parameters:

- `contentId` (string) – Video id to open

Returns: *Command*

launcher.launchNetflix (*contentId*) Launch Netflix app. Will launch deep-linked to provided `contentId`, if supported on the target platform.

On success, the success event/callback will be fired with the arguments (launchSession)

- `launchSession`: `LaunchSession`

Related capabilities:

- `Launcher.Netflix`
- `Launcher.Netflix.Params` – if launching with `contentId`

Parameters:

- `contentId` (string) – Video id to open

Returns: *Command*

launcher.launchYouTube (*contentId*) Launch YouTube app. Will launch deep-linked to provided `contentId`, if supported on the target platform.

On success, the success event/callback will be fired with the arguments (launchSession)

- `launchSession`: `LaunchSession`

Related capabilities:

- `Launcher.YouTube`
- `Launcher.YouTube.Params` – if launching with `contentId`

Parameters:

- `contentId` (string) – Video id to open

Returns: *Command*

launcher.getAppList () Gets a list of all apps installed on the device.

On success, the success event/callback will be fired with the arguments (appList)

- `appList`: `AppInfo[]` – Each `AppInfo` object contains:
 - `id` (string): platform-specific `appId`
 - `name` (string): human-readable name of app

Related capabilities:

- `Launcher.App.List`

Returns: *Command*

MediaControl

The `MediaControl` capability protocol serves to define the methods required for normalizing the control of media playback (play, pause, fast forward, etc) as well as obtaining media information (playhead position, duration, etc).

Methods

mediaControl.play () Send play command.

Related capabilities:

- `MediaControl.Play`

Returns: *Command*

mediaControl.pause () Send pause command.

Related capabilities:

- `MediaControl.Pause`

Returns: *Command*

mediaControl.stop () Send play command.

Related capabilities:

- `MediaControl.Stop`

Returns: *Command*

mediaControl.rewind () Send rewind command.

Related capabilities:

- `MediaControl.Rewind`

Returns: *Command*

mediaControl.fastForward () Send play command.

Related capabilities:

- `MediaControl.FastForward`

Returns: *Command*

mediaControl.seek (*position*) Seeks to a new position within the current media item

Related capabilities:

- `MediaControl.Seek`

Parameters:

- *position* (number) – Media seek position in seconds

Returns: *Command*

mediaControl.getDuration () On success, the success event/callback will be fired with the arguments (duration)

- *duration*: number – duration in seconds

Returns: *Command*

mediaControl.getPosition () On success, the success event/callback will be fired with the arguments (position)

- *position*: number – position in seconds

Returns: *Command*

mediaControl.subscribePlayState () On success, the success event/callback will be fired with the arguments (playState)

- *playState*: string – One of:

- “unknown”
- “idle”
- “playing”
- “paused”
- “buffering”
- “finished”

Returns: *Command*

MediaPlayer

The MediaPlayer capability protocol serves to define the methods required for displaying media on the device.

Methods

mediaPlayer.displayImage (*url*, *mimeType*, [*options*]) Display an image on the device. Not all devices support all of the parameters – supply as many as you have available.

On success, the success event/callback will be fired with the arguments (launchSession, mediaControl)

- launchSession: LaunchSession
- mediaControl: MediaControl

Related capabilities:

- MediaPlayer.Display.Image
- MediaPlayer.MediaData.Title
- MediaPlayer.MediaData.Description
- MediaPlayer.MediaData.Thumbnail
- MediaPlayer.MediaData.MimeType

Parameters:

- url (string)
- mimeType (string) – MIME type of the image, for example “image/jpeg”
- options (object) [optional] – All properties are optional:
 - title (string): Title text to display
 - description (string): Description text to display
 - iconUrl (string): URL of icon to show next to the title

Returns: *Command*

mediaPlayer.playMedia (*url*, *mimeType*, [*options*]) Play an audio or video file on the device. Not all devices support all of the parameters – supply as many as you have available.

On success, the success event/callback will be fired with the arguments (launchSession, mediaControl)

- launchSession: LaunchSession
- mediaControl: MediaControl

Related capabilities:

- `MediaPlayer.Play.Video`
- `MediaPlayer.Play.Audio`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- `url` (string)
- `mimeType` (string) – MIME type of the video, for example “video/mpeg4”, “audio/mp3”, etc
- `options` (object) [optional] – All properties are optional:
 - `title` (string): Title text to display
 - `description` (string): Description paragraph to display
 - `iconUrl` (string): URL of icon to show next to the title
 - `shouldLoop` (boolean): Whether to automatically loop playback
 - `subtitles` {object} subtitle track with options (properties are optional unless specified otherwise):
 - * `url` (string) [required]: must be a valid URL
 - * `mimeType` (string)
 - * `language` (string)
 - * `label` (string)

Returns: *Command*

MouseControl

The `MouseControl` capability serves to define the methods required for normalizing a mouse/trackpad (move/scroll with relative coordinates and click).

Methods

`mouseControl.connectMouse ()` Establish a connection with the `DeviceService`’s mouse communication medium (WebSocket, HTTP, etc). While this step may not be necessary with certain platforms, it is suggested to call it anyways, for purposes of seamless normalization. Calling `connect` on a non-connectable protocol will just trigger the success callback immediately.

Related capabilities:

- `MouseControl.Connect`

Returns: *Command*

`mouseControl.disconnectMouse ()` Disconnects from the mouse communication medium.

Related capabilities:

- `MouseControl.Disconnect`

Returns: *Command*

mouseControl.move (*dx*, *dy*) Move the mouse by the given distance values.

Related capabilities:

- `MouseControl.Move`

Parameters:

- *dx* (number) – Distance to move the mouse on the x-axis relative to its current position
- *dy* (number) – Distance to move the mouse on the y-axis relative to its current position

Returns: *Command*

mouseControl.scroll (*dx*, *dy*) Scroll by the given distance values.

Related capabilities:

- `MouseControl.Scroll`

Parameters:

- *dx* (number) – Distance to scroll the mouse on the x-axis relative to its current position
- *dy* (number) – Distance to scroll the mouse on the y-axis relative to its current position

Returns: *Command*

mouseControl.click () Perform a click action at the current mouse position.

Related capabilities:

- `MouseControl.Click`

Returns: *Command*

PlaylistControl

Methods

playlistControl.next () Jump playlist to the next track.

Related capabilities:

- `PlaylistControl.Next`

Returns: *Command*

playlistControl.previous () Jump playlist to the previous track.

Related capabilities:

- `PlaylistControl.Previous`

Returns: *Command*

playlistControl.jumpToTrack (*index*) Jump the playlist to the designated track.

Related capabilities:

- `PlaylistControl.JumpToTrack`

Parameters:

- *index* (number) – Playlist track index

Returns: *Command*

PowerControl

The PowerControl capability protocol serves to define the methods required for normalizing power off functionality.

Methods

powerControl.powerOff () Sends a power off signal to the TV. A success message will, internally, trigger a disconnection with the device.

Related capabilities:

- `PowerControl.Off`

Returns: *Command*

TVControl

The TVControl capability protocol serves to define the methods required for normalizing common TV-specific commands (channel up/down, channel list, channel info, etc).

ChannelInfo objects are plain JavaScript objects with the following properties:

- `id` (string): A platform-specific id used to identify the channel
- `name` (string): A human-readable name of the channel, if available
- `number` (string): Channel number such as “54-1”
- `majorNumber` (number): Major channel number
- `minorNumber` (number): Minor channel number (subchannel number)

Methods

tvControl.channelUp () Sends a channel up command to the TV.

Related capabilities:

- `TVControl.Channel.Up`

Returns: *Command*

tvControl.channelDown () Sends a channel down command to the TV.

Related capabilities:

- `TVControl.Channel.Down`

Returns: *Command*

tvControl.setChannel (*channelInfo*) Sets the current channel to the channel provided by the ChannelInfo object provided.

Related capabilities:

- `TVControl.Channel.Set`

Parameters:

- `channelInfo` (object) – ChannelInfo object containing information about the desired channel

Returns: *Command*

tvControl.getChannelList () Get a list of available channels from the TV.

On success, the success event/callback will be fired with the arguments (`channelInfoList`)

- `channelInfoList`: ChannelInfo[]

Related capabilities:

- `TVControl.Channel.List`

Returns: *Command*

tvControl.getCurrentChannel () Gets the current channel info from the TV.

On success, the success event/callback will be fired with the arguments (`channelInfo`)

- `channelInfo`: ChannelInfo

Related capabilities:

- `TVControl.Channel.Get`

Returns: *Command*

tvControl.subscribeCurrentChannel () Subscribes to any changes in the current channel. Each time the channel is changed, the new channel's info will be provided to the success callback.

On success, the success event/callback will be fired with the arguments (`channelInfo`)

- `channelInfo`: ChannelInfo

Related capabilities:

- `TVControl.Channel.Subscribe`

Returns: *Subscription*

TextInputControl

The `TextInputControl` capability serves to define the methods required for normalizing common text input commands (send text, enter, delete, keyboard status).

Methods

textInputControl.sendText (*input*) Send text to the current text field.

Related capabilities:

- `TextInputControl.Send.Text`

Parameters:

- `input` (string)

Returns: *Command*

textInputControl.sendEnter () Send enter key to the current text field.

Related capabilities:

- `TextInputControl.Send.Enter`

Returns: *Command*

textInputControl.sendDelete () Send delete event to the current text field.

Related capabilities:

- `TextInputControl.Send.Delete`

Returns: *Command*

textInputControl.subscribeTextInputStatus () Subscribe to information about the current text field.

On success, the success event/callback will be fired with the arguments (`textInputStatus`)

- `textInputStatus: TextInputStatus`

Related capabilities:

- `TextInputControl.Subscribe`

Returns: *Subscription*

ToastControl

The `ToastControl` capability protocol serves to define the methods required for displaying toast messages on the TV.

Toasts may optionally provide an 80x80 pixel icon in PNG or JPEG format, encoded as base64. The icon will be displayed alongside the toast message.

Methods

toastControl.showToast (*message*, [*options*]) Show a toast on the TV.

Parameters:

- `message` (string) – Message to display
Message to display
- `options` (object) [optional] –
 - `iconData` (string): base64-encoded image
 - `iconExtension` (string): file extension of icon (.png or .jpg)

Returns: *Command*

toastControl.showClickableToast (*message*, *options*) Show a toast on the TV and perform an action when the toast is clicked on the TV.

Related capabilities:

- `ToastControl.Show.Clickable.App`
- `ToastControl.Show.Clickable.App.Params`
- `ToastControl.Show.Clickable.URL`

Parameters:

- `message` (string) – Message to display
Message to display
- `options` (object) –

- iconData (string): base64-encoded image
- iconExtension (string): file extension of icon (.png or .jpg)
- appId (string): app to launch when clicked OR
- url (string): url to launch in browser when clicked

Returns: *Command*

VolumeControl

The VolumeControl capability protocol serves to define the methods required for normalizing common volume specific commands (volume up/down, mute, etc).

Methods

volumeControl.getVolume () Get the current volume of the device.

On success, the success event/callback will be fired with the arguments (volume)

- volume: number

Related capabilities:

- VolumeControl.Get

Returns: *Command*

volumeControl.setVolume (volume) Set the volume of the device.

Related capabilities:

- VolumeControl.Set

Parameters:

- volume (float) – Volume as a float between 0.0 and 1.0

Returns: *Command*

volumeControl.volumeUp () Sends the volume up command to the device.

Related capabilities:

- VolumeControl.UpDown

Returns: *Command*

volumeControl.volumeDown () Sends the volume down command to the device.

Related capabilities:

- VolumeControl.UpDown

Returns: *Command*

volumeControl.getMute () Get the current mute state.

On success, the success event/callback will be fired with the arguments (mute)

- mute: boolean

Related capabilities:

- VolumeControl.Mute.Get

Returns: *Command*

volumeControl.setMute (*mute*) Set the current volume.

Related capabilities:

- `VolumeControl.Mute.Set`

Parameters:

- `mute` (boolean)

Returns: *Command*

volumeControl.subscribeMute () Subscribe to the mute state on the TV.

On success, the success event/callback will be fired with the arguments (`mute`)

- `mute`: boolean

Related capabilities:

- `VolumeControl.Mute.Subscribe`

Returns: *Subscription*

volumeControl.subscribeVolume () Subscribe to the volume on the TV.

On success, the success event/callback will be fired with the arguments (`volume`)

- `volume`: number

Related capabilities:

- `VolumeControl.Subscribe`

Returns: *Subscription*

WebAppLauncher

The WebAppLauncher capability protocol provides capabilities for launching web apps and establishing two-way communication.

Methods

webAppLauncher.launchWebApp (*webAppId, params*) Launch a web application on the TV.

See WebAppSession for a detailed example.

On success, the success event/callback will be fired with the arguments (`webAppSession`)

- `webAppSession`: WebAppSession

Related capabilities:

- `WebAppLauncher.Launch`
- `WebAppLauncher.Launch.Params` – if launching with params

Parameters:

- `webAppId` (string) – ID of web app assigned by platform vendor
- `params` (object) – Dictionary of key/value strings. Not available on all target platforms

Returns: *Command*

webAppLauncher.joinWebApp (*webAppId, params*) Join an active web app without launching/relaunching. If the app is not running/joinable, the failure block will be called immediately.

On success, the success event/callback will be fired with the arguments (webAppSession)

- webAppSession: WebAppSession

Related capabilities:

- WebAppLauncher.Send
- WebAppLauncher.Receive

Parameters:

- webAppId (string) – Unique identifier for the web app to be joined
- params (object)

Returns: *Command*

webAppLauncher.closeWebApp (*webAppId*) Closes a web app with the provided LaunchSession.

Related capabilities:

- WebAppLauncher.Close

Parameters:

- webAppId (string)

Returns: *Command*

webAppLauncher.pinWebApp (*webAppId*) **Parameters:**

- webAppId (string)

Returns: *Command*

webAppLauncher.unPinWebApp (*webAppId*) **Parameters:**

- webAppId (string)

Returns: *Command*

webAppLauncher.isWebAppPinned (*webAppId*) **Parameters:**

- webAppId (string)

Returns: *Command*

webAppLauncher.subscribeIsWebAppPinned (*webAppId*) **Parameters:**

- webAppId (string)

Returns: *Command*

5.13.5 Constants

AirPlayServiceMode

Properties

WEBAPP display media using a web app mirrored to the TV (iOS only)

MEDIA display media using AirPlay media playback APIs

KeyCodes

Properties

NUM_0 NUM_1 NUM_2 NUM_3 NUM_4 NUM_5 NUM_6 NUM_7 NUM_8 NUM_9 DASH ENTER

PairingLevel

Properties

ON access to capabilities that require pairing

OFF access to capabilities that don't require pairing

PairingType

Properties

NONE Only connect if no pairing is required

FIRST_SCREEN Prompt the user on the TV to accept pairing

PIN Display a PIN on the TV, require user to enter it on the device

MIXED Prompt the user on the TV to accept pairing. Also display a pin on the TV that the user can enter on the device.

AIRPLAY_MIRRORING Require AirPlay mirroring to be enabled for connection (iOS only)

Services

Properties

Chromecast Chromecast

DIAL DIAL

DLNA DLNA

NetcastTV LG 2012/2013 Smart TV with Netcast

Roku Roku

WebOSTV LG 2014 Smart TV with webOS

FireTV Amazon FireTV

AirPlay Apple AirPlay

5.14 Getting Started

5.14.1 Modularization

Structure

The **Connect SDK repositories** are adopting a modular approach with 1.4.0 release. Our aim is to provide flexibility to the developers to be able pick and choose between the various devices. Currently you can choose whether to include **Google Cast** and **Fire TV** devices or not. We plan to include more device options in the upcoming releases.

The Connect SDK is split into modules with the help of **git submodules**. There are two options:

1. The **full** project (*Connect-SDK-iOS* and *Connect-SDK-Android*) includes three submodules: core, google-cast, and firetv and thus provides the full feature set. The latter submodules are located in the modules directory.
2. The **lite** project (*Connect-SDK-iOS-Lite* and *Connect-SDK-Android-Lite*) includes the core submodule only, therefore there is no need to download any third-party dependencies.

Please refer to the figure below displaying dependencies between different modules and libraries (for iOS and Android).

Components with a light green background are external dependencies. The dashed lines show the submodule links, whereas the solid lines depict build and/or runtime dependencies.

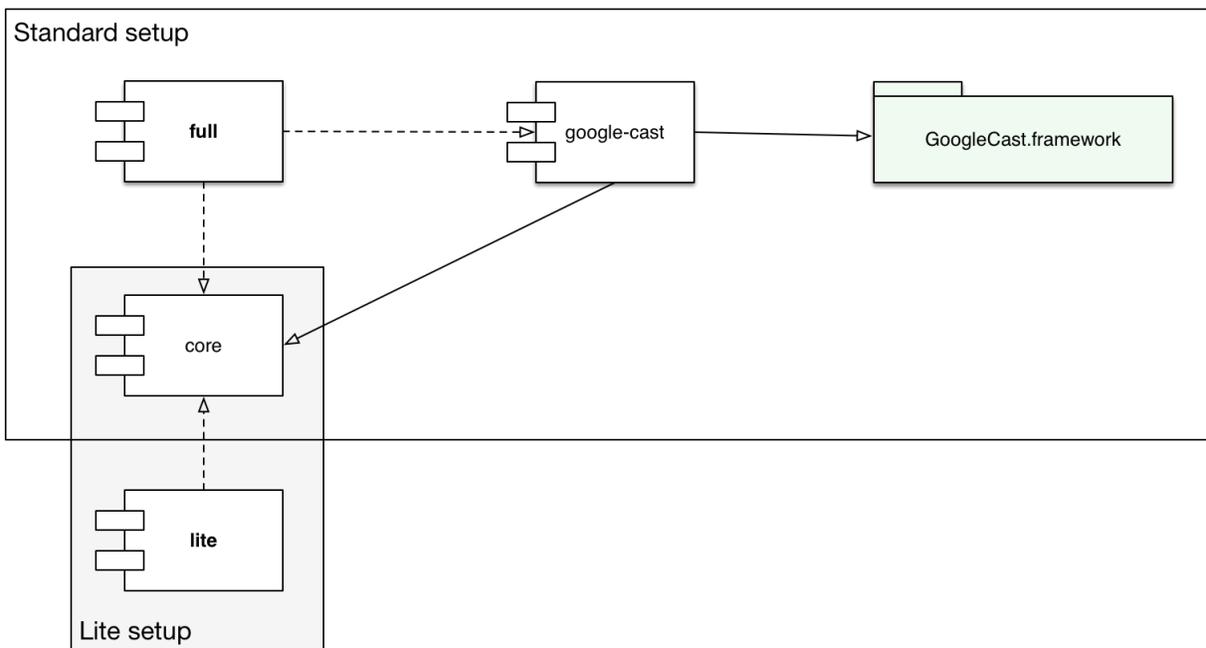


Fig. 2: Figure 1. iOS SDK Component Diagram (showing Google Cast submodule as an example)

Links to the repositories are provided in the next table:

Table 2: Table 1. Links to the repositories of iOS

Module	Link
full	https://github.com/ConnectSDK/Connect-SDK-iOS
lite	https://github.com/ConnectSDK/Connect-SDK-iOS-Lite
core	https://github.com/ConnectSDK/Connect-SDK-iOS-Core
google-cast	https://github.com/ConnectSDK/Connect-SDK-iOS-Google-Cast
firetv	https://github.com/ConnectSDK/Connect-SDK-iOS-FireTV

Usage instructions can be found in the **full README** or **lite README**.

Contributing

Since the source code is split between three repositories now (in the full version, whereas lite has only two), contributing is a bit more involved now. If you add a new feature across all the modules, you will have to create two GitHub pull requests, one for each module. Our team will check the code and merge the changes into the submodules, then update the full and lite repositories (as those just keep the project and track commits from the submodules). If you have a simpler contributing workflow in mind, please [let us know](#).

5.14.2 Setup Instructions

Using CocoaPods

1. Add `pod "ConnectSDK"` to your Podfile
2. Run `pod install`
3. Open the workspace file and run your project

Important: Unfortunately, Amazon Flaming SDK is not distributed via CocoaPods, so we cannot include its support in a subspec in an automated way. If you need it, please use the source ConnectSDK project directly.

You can use `pod "ConnectSDK/Core"` to get the [lite version](#).

Without CocoaPods

1. Clone the repository (`git clone https://github.com/ConnectSDK/Connect-SDK-iOS.git`)
2. Set up the submodules by running the following command (in the `Connect-SDK-iOS/` directory in this example): `git submodule update --init`
3. Open your project in Xcode
4. Locate the Connect SDK Xcode project in Finder
5. Drag the Connect SDK Xcode project (`ConnectSDK.xcodeproj`) into your project's Xcode library
6. Navigate to your target's settings screen, then navigate to the "Build Phases" tab
7. Add the following in the "Link Binary With Libraries" section:
 - `libConnectSDK.a`
 - `libz.dylib`
 - `libcucore.dylib`
8. Navigate to the "Build Settings" tab and add `-ObjC` to your target's "Other Linker Flags"
9. Follow the setup instructions for the service submodules:
 - [Connect-SDK-iOS-Google-Cast](#)
 - [Connect-SDK-iOS-FireTV](#)

If these steps are failing, try checking the [repository](#) for the latest setup instructions.

Include Strings File for Localization (optional)

1. Locate the Connect SDK Xcode project in the Finder
2. Drag the ConnectSDKStrings folder into your project's Resources folder
3. You may make whatever changes you would like to the values and the SDK will use your strings file

5.14.3 Discover & Connect to Device

Initial setup

Your view controller should implement delegate/listener methods for Connect SDK's DevicePicker and ConnectableDevice classes. These methods will give you the ability to respond to device selection, ready, disconnect, and error states.

```
@interface ViewController () <DevicePickerDelegate, ConnectableDeviceDelegate>
@end
```

It is helpful to retain local references to both the DiscoveryManager and the ConnectableDevice objects. In most use cases, these two classes will serve to provide most of the functionality required.

```
@implementation ViewController
{
    DiscoveryManager *_discoveryManager;
    ConnectableDevice *_device;
}
```

As soon as your app loads, you should instantiate the DiscoveryManager singleton and start discovery. As different devices can take a wide range of time to be discovered, it is recommended that discovery start as soon as possible after app launch.

```
- (void) viewDidLoad
{
    [super viewDidLoad];

    // This step could even happen in your app's delegate
    _discoveryManager = [DiscoveryManager sharedManager];
    [_discoveryManager startDiscovery];
}
```

Discovery & device selection

In many cases, your user will want to select one device from a list of many. You should present the DevicePicker to the user to receive their selection. The DevicePicker includes a dynamic listing of all devices that have been discovered on the network.

Passing the “sender” property of an IBAction will allow the SDK to present a popover view from a UIView if the user is on an iPad.

```
- (IBAction) hShareImage: (id) sender
{
    _discoveryManager.devicePicker.delegate = self;
    [_discoveryManager.devicePicker showPicker:sender];
}
```

Once the user has selected a device, you should immediately register for events from that device and then call the connect method.

```
- (void)devicePicker:(DevicePicker *)picker didSelectDevice:(ConnectableDevice_
↳*) device
{
    _device = device;
    _device.delegate = self;
    [_device connect];
}
```

Capability Filtering

If your app is making use of certain device capabilities (media playback/controls, web app launching, etc), it is strongly recommended that you create filters with this information for DiscoveryManager.

Devices that are discovered & shown in the picker will be guaranteed to have the set of capabilities that you have provided. This will prevent your users from selecting a device that has not yet acquired all of its protocols.

```
NSArray *videoCapabilities = @[
    kMediaPlayerDisplayVideo,
    kMediaControlAny,
    kVolumeControlVolumeUpDown
];

NSArray *imageCapabilities = @[
    kMediaPlayerDisplayImage
];

CapabilityFilter *videoFilter = [CapabilityFilter_
↳filterWithCapabilities:videoCapabilities];
CapabilityFilter *imageFilter = [CapabilityFilter_
↳filterWithCapabilities:imageCapabilities];

[[DiscoveryManager sharedManager] setCapabilityFilters:@[videoFilter, imageFilter]];
```

Check out the article on *capabilities* for more depth on this topic.

5.15 Developer Guides

5.15.1 Beam Media

A common use case with Connect SDK is to beam a simple media file (image, video, audio) to a TV. The following is a quick example of how you can beam an image onto a TV. This example assumes that you have discovered and connected to a device.

Beam an image file

```
NSURL *mediaURL = [NSURL URLWithString:@"http://www.connectsdk.com/files/9613/9656/
↳8539/test_image.jpg"]; // credit: Blender Foundation/CC By 3.0
NSURL *iconURL = [NSURL URLWithString:@"http://www.connectsdk.com/files/2013/9656/
↳8845/test_image_icon.jpg"]; // credit: sintel-durian.deviantart.com
```

(continues on next page)

(continued from previous page)

```

NSString *title = @"Sintel Character Design";
NSString *description = @"Blender Open Movie Project";
NSString *mimeType = @"image/jpeg";

MediaInfo *mediaInfo = [[MediaInfo alloc] initWithURL:mediaURL mimeType:mimeType];
mediaInfo.title = title;
mediaInfo.description = description;
ImageInfo *imageInfo = [[ImageInfo alloc] initWithURL:iconURL type:ImageTypeThumb];
[mediaInfo addImage:imageInfo];

__block MediaLaunchObject *launchObject;

[self.device.mediaPlayer displayImageWithMediaInfo:mediaInfo
                                     success:
^ (MediaLaunchObject *mediaLaunchObject) {
    NSLog(@"display photo success");

    // save the object reference to control media playback
    launchObject = mediaLaunchObject;

    // enable your media control UI elements here
}
                                     failure:
^ (NSError *error) {
    NSLog(@"display photo failure: %@", error.localizedDescription);
}]];

```

Beam an audio/video file

```

NSURL *mediaURL = [NSURL URLWithString:@"http://www.connectsdk.com/files/8913/9657/
↳0225/test_video.mp4"]; // credit: Blender Foundation/CC By 3.0
NSURL *iconURL = [NSURL URLWithString:@"http://www.connectsdk.com/files/7313/9657/
↳0225/test_video_icon.jpg"]; // credit: sintel-durian.deviantart.com
NSString *title = @"Sintel Trailer";
NSString *description = @"Blender Open Movie Project";
NSString *mimeType = @"video/mp4"; // audio/* for audio files

MediaInfo *mediaInfo = [[MediaInfo alloc] initWithURL:mediaURL mimeType:mimeType];
mediaInfo.title = title;
mediaInfo.description = description;
ImageInfo *imageInfo = [[ImageInfo alloc] initWithURL:iconURL type:ImageTypeThumb];
[mediaInfo addImage:imageInfo];

if ([self.device hasCapability:kMediaPlayerSubtitleWebVTT]) {
    NSURL *subtitlesURL = [NSURL URLWithString:@"http://ec2-54-201-108-205.us-west-2.
↳compute.amazonaws.com/samples/media/sintel_en.vtt"];
    SubtitleInfo *subtitleInfo = [SubtitleInfo infoWithURL:subtitlesURL
                                     andBlock:^(SubtitleInfoBuilder_
↳*builder) {
        builder.mimeType = @"text/vtt";
        builder.language = @"English";
        builder.label = @"English_
↳Subtitles";
    }];
    mediaInfo.subtitleInfo = subtitleInfo;
}

```

(continues on next page)

(continued from previous page)

```

}

__block MediaLaunchObject *launchObject;

[self.device.mediaPlayer playMediaWithMediaInfo:mediaInfo
                        shouldLoop:NO
                        success:
^ (MediaLaunchObject *mediaLaunchObject) {
    NSLog(@"play video success");

    // save the object reference to control media playback
    launchObject = mediaLaunchObject;

    // enable your media control UI elements here
}

                                failure:
^ (NSError *error) {
    NSLog(@"play video failure: %@", error.localizedDescription);
}]];

```

Control media playback

In the previous example, you will notice that the success block was called with a mediaControl object. In order to control the media in the current playback session, you will need to store a reference to this mediaControl object and call control methods on that object.

```

// pause media file
[launchObject.mediaControl pauseWithSuccess:nil failure:nil];

// play media file
[launchObject.mediaControl playWithSuccess:nil failure:nil];

// seek to 10 seconds
[launchObject.mediaControl seek:10 success:nil failure:nil];

// close media file
[launchObject.session closeWithSuccess:nil failure:nil];
// or
[self.device.mediaPlayer closeMedia:launchObject.session success:nil failure:nil];

```

Beam a playlist

```

NSURL *mediaURL = [NSURL URLWithString:@"your-playlist.m3u"];
NSURL *iconURL = [NSURL URLWithString:@"http://www.connectsdk.com/files/2013/9656/
↪8845/test_image_icon.jpg"]; // credit: sintel-durian.deviantart.com
NSString *title = @"Playlist";
NSString *description = @"Playlist description";
NSString *mimeType = @"application/x-mpegurl";

MediaInfo *mediaInfo = [[MediaInfo alloc] initWithURL:mediaURL mimeType:mimeType];
mediaInfo.title = title;
mediaInfo.description = description;
ImageInfo *imageInfo = [[ImageInfo alloc] initWithURL:iconURL type:ImageTypeThumb];

```

(continues on next page)

```
[mediaInfo addImage:imageInfo];

__block MediaLaunchObject *launchObject;

[self.device.mediaPlayer playMediaWithMediaInfo:mediaInfo
                        shouldLoop:NO
                        success:
^ (MediaLaunchObject *mediaLaunchObject) {
    // save the object reference to control playlist and media playback
    launchObject = mediaLaunchObject;

    // enable your media control UI elements here
}
                        failure:
^ (NSError *error) {
    NSLog(@"play playlist failure: %@", error.localizedDescription);
}]];
```

Control a playlist

```
// play previous track
[launchObject.playlistControl playPreviousWithSuccess:nil failure:nil];
// play next track
[launchObject.playlistControl playNextWithSuccess:nil failure:nil];
// play a track specified by index (starts from zero)
[launchObject.playlistControl jumpToTrackWithIndex:0 success:nil failure:nil];
```

Note: For beaming media to AirPlay devices, you must set the *AirPlayServiceMode* to *AirPlayServiceModeMedia*. See the *API docs* for more information.

5.15.2 Beam Web Apps

There are several platforms available that support the launching of web apps. A web app is typically run on a temporary basis in a full-screen browser instance.

Web App IDs

Both webOS and Chromecast require a web app ID for API calls to launch and communicate with web apps. This web app ID is translated into your web app's URL on web app launch.

For information on creating a web app ID for webOS, please visit the [LG registration site](#).

To learn how to register for a Chromecast web app ID, visit [Google's app ID registration site](#).

Launch web app with identifier

Connect SDK currently supports web app launching on webOS, Chromecast, and Apple TV devices. Both webOS and Chromecast will translate a web app identifier into your web app's URL.

```

NSString *webAppId;

if ([_device serviceWithName:@"webOS TV"])
    webAppId = @"5G7328DE";
else if ([_device serviceWithName:@"Chromecast"])
    webAppId = @"3E5106AB";
else if ([_device serviceWithName:@"AirPlay"])
    webAppId = @"http://www.example.com/";

if (!webAppId)
    return;

[_device.webAppLauncher launchWebApp:webAppId success:^(WebAppSession *webAppSession)
↪{
    NSLog(@"web app launch success");
} failure:^(NSError *error) {
    NSLog(@"web app launch error: %@", error.localizedDescription);
}]];

```

Communicate with web app

Bi-directional communication with your web app is made extremely simple. Data can be sent and received strongly-typed as a string or a keyed set of values (JSON object).

```

WebAppSession *_webAppSession;

[_device.webAppLauncher launchWebApp:webAppId success:^(WebAppSession *webAppSession)
↪{
    NSLog(@"web app launch success");

    _webAppSession = webAppSession;
    _webAppSession.delegate = self;

    [_webAppSession connectWithSuccess:^(id responseObject) {
        NSLog(@"web app connect success");
    } failure:^(NSError *error) {
        NSLog(@"web app connect error: %@", error.localizedDescription);
    }]];
} failure:^(NSError *error) {
    NSLog(@"web app launch error: %@", error.localizedDescription);
}]];

```

After successfully establishing a connection, you can send messages to your web app.

```

[_webAppSession sendText:@"This is a test message" success:nil failure:nil];

```

You can also send an NSDictionary which will be received by the web app as a JSON object.

```

NSDictionary *message = @{
    @"someParameter" : @"someValue",
    @"anArray": @[
        @"array value 1",
        @"array value 2",
        @"array value 3"
    ],
    @"anotherObject" : @{

```

(continues on next page)

(continued from previous page)

```

        @"anotherParameter" : @"anotherValue"
    }
};

[_webAppSession sendJSON:message success:nil failure:nil];

```

WebAppSessionDelegate allows you to receive messages from your web app.

```

<code>:
- (void) webAppSession:(WebAppSession *)webAppSession didReceiveMessage:(id)message {
    // message may be either an NSString or an NSDictionary, depending on what was
    ↪ sent from the web app
    NSLog(@"Received message from web app %@", message);
}

```

Beam media to web app

A common use case for web apps is the playback and control of media files. Connect SDK provides capabilities for directly playing/controlling media on a WebAppSession, provided that web app has integrated the *Connect SDK JavaScript Bridge*.

Rather than calling playMedia on your device's mediaPlayer, webAppSession provides its own mediaPlayer. After media has been beamed into the web app, the control is just like any other media session.

```

MediaInfo *mediaInfo = [[MediaInfo alloc] initWithURL:mediaURL mimeType:mimeType];
mediaInfo.title = title;
mediaInfo.description = description;
ImageInfo *imageInfo = [[ImageInfo alloc] initWithURL:iconURL type:ImageTypeThumb];
[mediaInfo addImage:imageInfo];

[webAppSession.mediaPlayer playMediaWithMediaInfo:mediaInfo
                          shouldLoop:NO
                          success:
^ (MediaLaunchObject *mediaLaunchObject) {
    NSLog(@"play video success");

    // save the object reference to control media playback
    launchObject = mediaLaunchObject;

    // enable your media control UI elements here
}
                          failure:
^ (NSError *error) {
    NSLog(@"play video failure: %@", error.localizedDescription);
}
];

```

Note: For beaming media to AirPlay devices, you must set the *AirPlayServiceMode* to *AirPlayServiceModeMedia*. See the *API docs* for more information.

5.15.3 Launch App on TV

Many TVs and streaming players include support for launching installed apps. The following is a simplified example of how to launch YouTube on a device.

Launch an app

```
[_device.launcher launchApp:@"YouTube" success:^(LaunchSession *launchSession) {
    NSLog(@"app launch success");
} failure:^(NSError *error) {
    NSLog(@"app launch error: %@", error.localizedDescription);
}];
```

Device-specific app identifiers

On each device (webOS TV, Roku, etc) apps are identified by different values. Here is an example of the different identifiers in use for the YouTube app.

- webOS: youtube.leanback.v4 (value may change with future updates)
- Netcast: 0000000000017498 (value may be different on each TV)
- DIAL: YouTube (listed in [DIAL registry](#))
- Roku: 837 (Roku-specific channel number)

Launching an app with device-specific identifiers

The following snippet shows how to detect the platform of your device and launch with the appropriate app identifier.

```
NSString *appId;

if ([_device serviceWithName:@"webOS TV"])
    appId = @"youtube.leanback.v4";
else if ([_device serviceWithName:@"Netcast TV"])
    appId = @"0000000000017498";
else if ([_device serviceWithName:@"Roku"])
    appId = @"837";
else if ([_device serviceWithName:@"DIAL"])
    appId = @"YouTube";

if (!appId)
    return;

AppInfo *appInfo = [AppInfo appInfoForId:appId];
appInfo.name = @"YouTube";

[_device.launcher launchAppWithInfo:appInfo success:^(LaunchSession *launchSession) {
    NSLog(@"app launch success");
} failure:^(NSError *error) {
    NSLog(@"app launch error: %@", error.localizedDescription);
}];
```

AppInfo helper object

You will notice that the previous example refers to an AppInfo object. This object is used internally by Connect SDK to manage an app's protocol-specific properties. If a device supports app list, the app list will return a set of AppInfo objects for each app installed on the TV.

Launching an app with parameters

In most cases, a device's launcher object will allow you to pass launch parameters to your app. Connect SDK has normalized the parameter input type to a keyed set of values. These values are then parsed into the appropriate format for the protocol (XML, JSON, URL params, etc).

```
NSMutableDictionary *params = @{
    @"someProperty" : @"someValue"
};

[_device.launcher launchAppWithInfo:appInfo params:params success:^(LaunchSession_
↳ *launchSession) {
    NSLog(@"app launch success");
} failure:^(NSError *error) {
    NSLog(@"app launch error: %@", error.localizedDescription);
}];
```

Note: Due to the variety of protocols in use, it is strongly recommended that you only use strings for the keys AND values of your parameters.

5.15.4 Discovery Manager

At the heart of Connect SDK is DiscoveryManager, a multi-protocol service discovery engine with a pluggable architecture. Much of your initial experience with Connect SDK will be with the DiscoveryManager class, as it consolidates discovered service information into ConnectableDevice objects.

DiscoveryManager supports discovering services of differing protocols by using DiscoveryProviders. Many services are discoverable over SSDP and are registered to be discovered with the SSDPDiscoveryProvider class.

As services are discovered on the network, the DiscoveryProviders will notify DiscoveryManager. DiscoveryManager is capable of attributing multiple services, if applicable, to a single ConnectableDevice instance. Thus, it is possible to have a mixed-mode ConnectableDevice object that is theoretically capable of more functionality than a single service can provide.

DiscoveryManager keeps a running list of all discovered devices and maintains a filtered list of devices that have satisfied any of your CapabilityFilters. This filtered list is used by the DevicePicker when presenting the user with a list of devices.

Connect SDK device discovery can be started in one line.

```
[[DiscoveryManager sharedManager] startDiscovery];
```

Features

Filtering devices by capability

It will be necessary in many cases to filter out devices that don't support a desired feature-set. `DiscoveryManager` provides the `setCapabilityFilters` method to provide for this ability.

Here is a simple example that discovers devices that support (video playback AND any media controls AND volume up/down) OR (image display).

```
NSArray *videoCapabilities = @[
    kMediaPlayerDisplayVideo,
    kMediaControlAny,
    kVolumeControlVolumeUpDown
];

NSArray *imageCapabilities = @[
    kMediaPlayerDisplayImage
];

CapabilityFilter *videoFilter = [CapabilityFilter
    ↪filterWithCapabilities:videoCapabilities];
CapabilityFilter *imageFilter = [CapabilityFilter
    ↪filterWithCapabilities:imageCapabilities];

[[DiscoveryManager sharedManager] setCapabilityFilters:@[videoFilter, imageFilter]];
```

DeviceService registration

By default, Connect SDK is configured to discover all the services that it supports (webOS, Netcast, Chromecast, AirPlay, DIAL, & Roku). It is possible to support only a subset of these services by manually registering those services before starting `DiscoveryManager` for the first time.

```
[[DiscoveryManager sharedManager] registerDeviceService:[AirPlayService class]
    ↪withDiscovery:[ZeroconfDiscoveryProvider class]];
[[DiscoveryManager sharedManager] registerDeviceService:[CastService class]
    ↪withDiscovery:[CastDiscoveryProvider class]];
[[DiscoveryManager sharedManager] registerDeviceService:[DIALService class]
    ↪withDiscovery:[SSDPDiscoveryProvider class]];
[[DiscoveryManager sharedManager] registerDeviceService:[RokuService class]
    ↪withDiscovery:[SSDPDiscoveryProvider class]];
[[DiscoveryManager sharedManager] registerDeviceService:[DLNATVService class]
    ↪withDiscovery:[SSDPDiscoveryProvider class]]; // LG TV devices only, includes
    ↪NetcastTVService
[[DiscoveryManager sharedManager] registerDeviceService:[WebOSTVService class]
    ↪withDiscovery:[SSDPDiscoveryProvider class]];
```

Automatic stop/resume on app state change

If `DiscoveryManager` is running while your app enters a background state, it will resume immediately upon returning to a foreground state. This is to prevent battery drain on the user's device.

Pairing level

Connect SDK has support for pairing with certain devices. To have pairing disabled may reduce the number of supported capabilities that a `ConnectableDevice` has. Certain devices, although they may support the features you are filtering for, may not pass your `CapabilityFilter` if pairing is disabled.

See the *Supported Features* list for information on what devices require pairing for certain capabilities.

For the best user experience, Connect SDK has disabled pairing by default. Pairing can be enabled very easily, but it must be enabled before `DiscoveryManager` is started for the first time.

```
[DiscoveryManager sharedManager].pairingLevel = DeviceServicePairingLevelOn;
```

Device store

When connecting with a device certain information is retained about that connection. This information is helpful for app relaunches, pairing, remembering commonly-used devices, and more. Connect SDK provides a `ConnectableDeviceStore` protocol to allow you to store `ConnectableDevice` information in a manner that suits your use case.

A default implementation, `DefaultConnectableDeviceStore`, will be used by `DiscoveryManager` if no other `ConnectableDeviceStore` is provided to `DiscoveryManager` when `startDiscovery` is called.

See also:

- *DiscoveryManager*
- *CapabilityFilter*
- *PairingLevel*
- *ConnectableDeviceStore*
- *DefaultConnectableDeviceStore*

5.15.5 Checking Capabilities

Setting up filters

When you are discovering devices you are able to specify multiple capability filters.

```
NSArray *videoCapabilities = @[
    kMediaPlayerDisplayVideo,
    kMediaControlAny,
    kVolumeControlVolumeUpDown
];

NSArray *imageCapabilities = @[
    kMediaPlayerDisplayImage
];

CapabilityFilter *videoFilter =
    [CapabilityFilter filterWithCapabilities:videoCapabilities];
CapabilityFilter *imageFilter =
    [CapabilityFilter filterWithCapabilities:imageCapabilities];

[[DiscoveryManager sharedManager] setCapabilityFilters:@[videoFilter, imageFilter]];
```

Any service that is found may meet the requirements of either filter but not both. When getting the UI ready if a device might have a capability you should always check before enabling that UI component.

```
[myImageButton setEnabled:[self.device hasCapability:kMediaPlayerDisplayImage]];
```

5.15.6 Resuming Apps

It may be necessary for your app to resume from a background or closed state and re-establish connection with a previously connected device. There are many ways in which Connect SDK provides information to allow for this behavior.

ConnectableDevice ID

Each ConnectableDevice has a unique ID assigned to it upon creation. When that device is connected to, the device store saves information about each of the device's services. The unique ID persists across app launches by attributing service UUIDs to the unique device ID in the device store.

LaunchSession

The ability to interact with an app requires some information to persist, including a session ID. This session ID may be required to close the app, as well as allow the app to accurately track certain state information.

WebAppSession

The ability to communicate with a web app requires a LaunchSession object and/or the web app id.

Resuming most recent connection

In order to save & reconnect to a previously connected device, all you need to keep track of is the device's ID. Assuming you are using the ConnectableDeviceStore included with Connect SDK, previously connected devices will persist the same ID between app launches.

When your app restarts, you should immediately start discovery and listen for device found events from DiscoveryManager. In these events, you can check each device's ID and call `connect` on the previously connected device.

Important note about reconnecting

Just because your device has been discovered on the network doesn't mean that all of its services/capabilities are available. You will need to set a CapabilityFilter on DiscoveryManager or manually check the ConnectableDevice's capabilities before you call `connect`.

Save device ID to disk

```
ConnectableDevice *device; // device you've connected to

[[NSUserDefaults standardUserDefaults] setObject:device.id forKey:@"recentDeviceId"];

// save right away before entering background
[[NSUserDefaults standardUserDefaults] synchronize];
```

Reconnect to device

```
ConnectableDevice *mDevice;
NSString *mRecentDeviceId;

- (void) viewDidLoad {
    [super viewDidLoad];

    mRecentDeviceId = [[NSUserDefaults standardUserDefaults] objectForKey:@"recentDeviceId"];

    [[DiscoveryManager sharedManager] setCapabilityFilters:myCapabilityFilters];
    [[DiscoveryManager sharedManager] setDelegate:self];
    [[DiscoveryManager sharedManager] start];
}

- (void) discoveryManager:(DiscoveryManager *)manager_
didFindDevice:(ConnectableDevice *)device {
    if (mRecentDeviceId && !mDevice) {
        if ([device.id isEqualToString:mRecentDeviceId]) {
            mDevice = device;
            [device setDelegate:self];
            [device connect];
        }
    }
}
```

Resuming a web app session

Resuming a web app session is as simple as saving the WebAppSession's LaunchSession object before entering the background. It can even be serialized into a JSON object for easy cross-platform storage.

Save session info to disk

```
WebAppSession *webAppSession; // retrieved from WebAppLauncher launch success block

LaunchSession *launchSession = webAppSession.launchSession;
NSDictionary *launchSessionInfo = [launchSession toJSONObject];

[[NSUserDefaults standardUserDefaults] setObject:launchSessionInfo forKey:@"launchSession"];

// save right away before entering background
[[NSUserDefaults standardUserDefaults] synchronize];
```

Re-create session after device is connected/ready

```
ConnectableDevice *device; // device that has been re-discovered & re-connected

NSDictionary *launchSessionInfo = (NSDictionary *) [[NSUserDefaults_
launchSessionInfo];
LaunchSession *launchSession = [LaunchSession_
launchSessionFromJSONObject:launchSessionInfo];
```

(continues on next page)

(continued from previous page)

```
[device.webAppLauncher joinWebApp:launchSession
                        success:^(WebAppSession *webAppSession) { }
                        failure:^(NSError *) { } ];
```

Low-effort re-connection option

Alternatively, you could re-join your web app with just the web app id. This could have the side effect of generating new session information for your user, which may not be desired.

```
[device.webAppLauncher joinWebAppWithId:@"your web app id"
                        success:^(WebAppSession *webAppSession) { }
                        failure:^(NSError *) { } ];
```

See also:

- [Discover & Connect to Device](#)
- [Checking Capabilities](#)
- [Beam Web Apps](#)

5.15.7 Screen Mirroring

With Connect SDK integrated in the mobile app, it can cast the screen and sound into the TV screen. This allows you to extend the screen of a mobile app to a larger TV screen and share it with your family. Screen mirroring is a way to display the entire app screen to the TV.

Note: This feature is only supported on webOS TV 22.

Requirements

Including the Connect SDK using CocoaPods and setting up for screen mirroring

Add pod "ConnectSDK" to your Podfile, and run `pod install`. Open the workspace file and run your project.

Note that screen mirroring runs on iOS 12 and higher. In case of Broadcast Upload Extension for Screen Mirroring, set the `APPLICATION_EXTENSION_API_ONLY` value to `NO`. Refer to the Podfile example below.

```
platform :ios, '12.0'

def app_pods
  pod 'ConnectSDK/Core', :git => 'https://github.com/ConnectSDK/Connect-SDK-iOS.git'
  ↳', :branch => 'master', :submodules => true
end

target 'ScreenMirroring-Sampler' do
  use_frameworks!
  app_pods
end
```

(continues on next page)

(continued from previous page)

```
target 'ScreenMirroring-Extension-Sampler' do
  use_frameworks!
  app_pods

  post_install do |installer|
    installer.pods_project.targets.each do |target|
      target.build_configurations.each do |config|
        config.build_settings['APPLICATION_EXTENSION_API_ONLY'] = 'No'
      end
    end
  end
end
```

ReplayKit - Broadcast Upload Extension

To capture iPhone screen, you need to implement Broadcast Upload Extension using Replay Kit. Refer to the link below.

- [AppleDeveloper - ReplayKit](#)
- [WWDC2020 Capture and stream apps on the Mac with ReplayKit](#)

How to Use Screen Mirroring

To use screen mirroring, follow these steps.

1. Search Devices

Search for devices (TVs) connected to your home network. You can set the filter to only search for TVs that support the screen mirroring function. Since the search for TVs takes some time, it should be started as soon as the app is running.

```
- (void)startDiscoveryTV {
    _isDiscoveringTV = YES;

    if (_discoveryManager == nil) {
        _discoveryManager = [DiscoveryManager sharedManager];
    }

    // Sets a device search filter (Screen Mirroring Capability) for devices that
    ↪support screen mirroring
    NSArray *capabilities = @[
        kScreenMirroringControlScreenMirroring
    ];

    CapabilityFilter *filter = [CapabilityFilter filterWithCapabilities:capabilities];
    [_discoveryManager setCapabilityFilters:@[filter]];
    [_discoveryManager setPairingLevel:DeviceServicePairingLevelOn];
    [_discoveryManager registerDeviceService:[WebOSTVService class]
    ↪withDiscovery:[SSDPDiscoveryProvider class]];
    [_discoveryManager startDiscovery];
}
```

2. Select a TV

Select the TV to run the screen mirroring on by using the Picker.

```
_discoveryManager.devicePicker.delegate = self;
[_discoveryManager.devicePicker showPicker:nil];
```

Once the user has selected a device, the application needs to store that device identifier to find it. This sample code uses `NSUserDefaults` to store its device identifier.

```
// MARK: DevicePickerDelegate
- (void)devicePicker:(DevicePicker *)picker didSelectDevice:(ConnectableDevice_
↳*)device {
    NSString *groupId = @"YOUR APP GROUP ID";
    NSUserDefaults *sharedDefaults = [[NSUserDefaults alloc]_
↳initWithSuiteName:groupId];
    [sharedDefaults setObject:device.address_
↳ forKey:kConnectableDeviceIpAddressKey];
    [sharedDefaults synchronize];
}
```

3. Start Screen Mirroring

Now you can run the screen mirroring. Start capturing the screen by creating an `RPSystemBroadcastPickerView`.

```
if (@available(iOS 12.0, *)) {
    RPSystemBroadcastPickerView *rpPickerView = [[RPSystemBroadcastPickerView alloc]_
↳initWithFrame:_rpPickerView.bounds];
    rpPickerView.preferredExtension = @"YOUR EXTENSION BUNDLE ID";
    rpPickerView.showsMicrophoneButton = NO;
    UIButton *button = rpPickerView.subviews.firstObject;
    button.imageView.tintColor = UIColor.whiteColor;
    [_rpPickerView addSubview:rpPickerView];
} else {
    /* UNAVAILABLE */
}
```

After the screen capture starts, you need to search once again with the information of selected TV device stored in the application.

```
- (instancetype)init {
    self = [super init];

    _discoveryManager = [DiscoveryManager sharedManager];

    NSString *groupId = @"YOUR APP GROUP ID";
    NSUserDefaults *sharedDefaults = [[NSUserDefaults alloc]_
↳initWithSuiteName:groupId];
    _deviceAddress = [sharedDefaults stringForKey:kConnectableDeviceIpAddressKey];

    NSArray *capabilities = @[ kScreenMirroringControlScreenMirroring ];
    CapabilityFilter *filter = [CapabilityFilter filterWithCapabilities:capabilities];
    [_discoveryManager setCapabilityFilters:@[filter]];
    [_discoveryManager setPairingLevel:DeviceServicePairingLevelOn];
    [_discoveryManager registerDeviceService:[WebOSTVService class]_
↳withDiscovery:[SSDPDiscoveryProvider class]];
}
```

(continues on next page)

(continued from previous page)

```

        [_discoveryManager startDiscovery];
        [_discoveryManager setDelegate:self];

        return self;
    }

```

If you find your TV again, get a `ScreenMirroringControl` object to use the screen mirroring API. And then, you should immediately call the `startScreenMirroring` method.

```

// MARK: DiscoveryManagerDelegate
- (void)discoveryManager:(DiscoveryManager *)manager didFindDevice:(ConnectableDevice_
↳*)device {
    if ([device.address caseInsensitiveCompare:_deviceAddress] != NSOrderedSame) {
        return;
    }

    _device = device;
    _screenMirroringControl = [_device screenMirroringControl];

    if (_screenMirroringControl != nil) {
        [_screenMirroringControl startScreenMirroring];
        [_screenMirroringControl setScreenMirroringDelegate:self];
    }

    [_discoveryManager stopDiscovery];
}

```

Handle Runtime Errors

The following runtime errors might occur while the screen mirroring is running.

- When the network connection is terminated
- When the TV is turned off
- When the screen mirroring is terminated on the TV
- When the mobile device's notification terminates the screen mirroring
- When other exceptions occurred

For these errors, it is necessary to receive the error in real-time through the listener and respond appropriately.

```

// MARK: ScreenMirroringControlDelegate
- (void)screenMirroringDidStart:(BOOL)result {
    NSLog(@"screenMirroringDidStart %d", result);
}

- (void)screenMirroringDidStop:(BOOL)result {
    NSLog(@"screenMirroringDidStop %d", result);
}

- (void)screenMirroringErrorDidOccur:(ScreenMirroringError)error {
    NSLog(@"screenMirroringErrorDidOccur %d", error);
    [self finishBroadcastWithError:NULL];
}

```

4. Broadcast Upload Extension Handling

You can get CMSampleBufferRef and RPSampleBufferType via SampleHandler's processSampleBuffer:withType:. It must be delivered to the screen mirroring API.

```
- (void)processSampleBuffer:(CMSampleBufferRef) sampleBuffer_
↳withType:(RPSampleBufferType) sampleBufferType {
    // Handle video sample buffer and audio sample buffer for app
    if (_screenMirroringControl != nil) {
        [_screenMirroringControl pushSampleBuffer:sampleBuffer with:sampleBufferType];
    }
}
```

5. Stop Screen Mirroring

When you want to stop mirroring, call stopScreenMirroring.

```
- (void)broadcastFinished {
    // User has requested to finish the broadcast.
    if (_screenMirroringControl != nil) {
        [_screenMirroringControl stopScreenMirroring];
    }
}
```

5.15.8 Remote Camera

With Connect SDK integrated in the mobile app, it can display camera preview on the TV screen. This allows you to use your mobile device's camera as a remote camera for the TV that does not have an internal or USB camera.

Note: This feature is only supported on webOS TV 22.

Requirements

Including the Connect SDK using CocoaPods and setting up for remote camera

Add pod "ConnectSDK" to your Podfile, and run pod install. Open the workspace file and run your project.

Note that remote camera runs on iOS 12 and higher. Refer to the Podfile example below.

```
platform :ios, '12.0'

def app_pods
  pod 'ConnectSDK/Core', :git => 'https://github.com/ConnectSDK/Connect-SDK-iOS.git'
  ↳', :branch => 'master', :submodules => true
end

target 'RemoteCamera-Sampler' do
  use_frameworks!
  app_pods
end
```

How to Use Remote Camera

To use a remote camera, follow the steps below.

1. Search Devices

Search for devices (TVs) connected to your home network. You can set the filter to only search for TVs that support the remote camera function.

```
...
- (void)startDiscoveryTV {
    _isDiscoveringTV = YES;

    if (_discoveryManager == nil) {
        _discoveryManager = [DiscoveryManager sharedManager];
    }

    NSArray *capabilities = @[
        kRemoteCameraControlRemoteCamera
    ];

    CapabilityFilter *filter = [CapabilityFilter filterWithCapabilities:capabilities];
    [_discoveryManager setCapabilityFilters:@[filter]];
    [_discoveryManager setPairingLevel:DeviceServicePairingLevelOn];
    [_discoveryManager registerDeviceService:[WebOSTVService class]
↳withDiscovery:[SSDPDiscoveryProvider class]];
    [_discoveryManager startDiscovery];
}
...
```

2. Request Permissions

The remote camera function requires the camera and microphone permission. The user must grant these permissions when the remote camera is first executed. Register NSCameraUsageDescription and NSMicrophoneUsageDescription in Info.plist.

```
<key>NSCameraUsageDescription</key>
<string></string>
<key>NSMicrophoneUsageDescription</key>
<string></string>
```

3. Select a TV

Select the TV to run the remote camera on by using the Picker. Implement DevicePickerDelegate to receive TV selection events.

```
_discoveryManager.devicePicker.delegate = self;
[_discoveryManager.devicePicker showPicker:nil];
```

Create a ViewController to display the camera preview after the TV is selected. You need to make ViewController work only in landscape mode.

```
// MARK: DevicePickerDelegate
- (void)devicePicker:(DevicePicker *)picker didSelectDevice:(ConnectableDevice_
↳*) device {
    RemoteCameraViewController *vc = [self.storyboard_
↳instantiateViewControllerWithIdentifier:@"RemoteCameraViewController"];
    [vc setDevice:device];
    [self presentViewController:vc animated:YES completion:nil];
}
```

Get a RemoteCameraControl object to use the remote camera API. And implement RemoteCameraControlDelegate to receive events that occur during remote camera operation.

```
_remoteCameraControl = [_device remoteCameraControl];
[_remoteCameraControl setRemoteCameraDelegate:self];
```

4. Start Remote Camera

Now you can run the remote camera. First, connect with the selected TV device through startRemoteCamera of RemoteCameraControl. Then show the camera preview in the returned UIView. Paring is required if this is the first time connecting to a TV.

```
UIView *previewView = [_remoteCameraControl startRemoteCamera];
[previewView setFrame:UIScreen.mainScreen.bounds];
[self.view addSubview:previewView];
[self.view sendSubviewToBack:previewView];
```

5. Start Camera Playback

Select iPhone camera on your TV. It will start sending and playing the camera stream. At this time, you can receive callbacks by designating a delegate.

```
// MARK: RemoteCameraControlDelegate
- (void)remoteCameraDidPlay {
    NSLog(@"remoteCameraDidPlay");
}

- (void)remoteCameraDidChange:(RemoteCameraProperty)property {
    NSLog(@"remoteCameraDidChange");
}
```

6. Stop Remote Camera

When you want to stop the remote camera, call stopRemoteCamera.

```
if (_remoteCameraControl != nil) {
    [_remoteCameraControl stopRemoteCamera];
    _remoteCameraControl = nil;
}
```

Features

Change Camera Property

You can change camera properties such as brightness and AWB on the TV, and you can receive callbacks by designating a delegate.

```
// MARK: RemoteCameraControlDelegate
- (void)remoteCameraDidChange:(RemoteCameraProperty)property {
    NSLog(@"remoteCameraDidChange");
}
```

Handle Runtime Errors

The following runtime error might occur while the remote camera is running.

- When the network connection is terminated
- When the TV is turned off
- When the remote camera is terminated on the TV
- When the mobile device's notification terminates the remote camera
- When other exceptions occurred

For these errors, it is necessary to receive the error in real-time through the listener and respond appropriately.

```
- (void)remoteCameraErrorDidOccur:(RemoteCameraError)error {
    NSLog(@"remoteCameraErrorDidOccur");

    if (_remoteCameraControl != nil) {
        [_remoteCameraControl stopRemoteCamera];
        _remoteCameraControl = nil;
    }
}
```

Also, if the app is in the background state, the remote camera function does not work, so you have to handle these situations appropriately.

```
- (void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];

    ...

    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(didEnterBackground)
                                             name:UIApplicationDidEnterBackgroundNotification object:nil];
}

- (void)didEnterBackground {
    if (_remoteCameraControl != nil) {
        [_remoteCameraControl stopRemoteCamera];
        _remoteCameraControl = nil;
    }
}
```

(continues on next page)

(continued from previous page)

```

- (void)viewWillDisappear:(BOOL)animated {
    [super viewWillDisappear:animated];

    [[NSNotificationCenter defaultCenter] addObserver:self
                                             name:UIApplicationDidEnterBackgroundNotification
                                             object:nil];
}

```

Set the Microphone Mute State

If you change the microphone mute state, it must be transmitted. The app must maintain the current mute setting value.

```

if (_remoteCameraControl != nil) {
    [_remoteCameraControl setMicMute:_isMuted];
}

```

Switch between Front and Back Cameras

When the direction of the camera is switched between front and rear, the camera direction is transmitted. The app must maintain the current camera direction value.

```

if (_remoteCameraControl != nil) {
    [_remoteCameraControl setLensFacing:lensFacing];
}

```

5.15.9 FAQ

When do I start the DiscoveryManager?

We recommend starting the DiscoveryManager when the app is started so that devices can be discovered and ready for use by the time the UI is loaded.

If you need to start the discovery later or only during a specific activity within your app you should be aware that it can take a few seconds for devices to be discovered.

How do I reconnect to a device on resume?

When your app goes into the background you can hold onto a ConnectableDevice object. When your app resumes you have the reference to the ConnectableDevice and you can listen for the Device ready function. Once the device is ready you can call connect and begin using it again.

How do I re-connect to a Web App when app resumes?

When a WebApp is launched on a TV you get a reference to that WebApp's WebAppSession object. When your phone's application goes into the background you can hold onto that WebAppSession object for the next time your application is in the foreground. Once your app is in the foreground again and you get a ConnectableDevice object.

```
connectableDeviceReady:
```

Then once the method is called you can use the stored `WebAppSession` object to continue to send commands to the running app.

How do I get the number of devices discovered?

When you start an app you should always assume that there are 0 devices discovered. Using the `DiscoveryManagerDelegate` you will be notified whenever a new device is discovered and an existing device has been lost.

```
discoveryManager:didFindDevice:  
discoveryManager:didLoseDevice:
```

When either of these methods are called you can reference the `compatibleDevices` property of the `sharedManager` to get a complete list of devices that match your filters.

When there are no compatible devices your UI should reflect this by hiding the beam icon.

How do create an ADHoc list of devices?

When you specify your device filters you may have devices that don't support every feature. If you are searching for all devices that can either display an image or play a YouTube video then you want to show a list of all the devices that can show an image.

To do this you will need to check that each device in the `compatibleDevices` array has the capabilities that you are looking for.

```
- (NSArray *) getImageDevices  
{  
    NSMutableArray *imageDevices = [NSMutableArray new];  
  
    for (ConnectableDevice *device in [DiscoveryManager sharedManager].  
↳compatibleDevices)  
    {  
        if ([device hasCapability:kMediaPlayerDisplayImage])  
            [imageDevices addObject:device];  
    }  
  
    return imageDevices;  
}
```

How do I show an image or video from my device?

All videos that are sent with the Connect SDK are links to external web content and your device is no different. You can setup a quick HTTP server and pass the url of your phone with Connect SDK. The media player will reach to your HTTP server and stream your content right from there.

There are some pre-made libraries that already do the heavy lifting for you.

Checkout: [CocoaHTTPServer](#)

5.16 API References

5.16.1 Discovery

CapabilityFilter

CapabilityFilter is an object that wraps an NSArray of required capabilities. This CapabilityFilter is used for determining which devices will appear in DiscoveryManager’s compatibleDevices array. The contents of a CapabilityFilter’s array must be any of the string constants defined in the Capability header files.

CapabilityFilter values

Here are some examples of values for the Capability constants.

- kMediaPlayerPlayVideo = “MediaPlayer.Display.Video”
- kMediaPlayerDisplayImage = “MediaPlayer.Display.Image”
- kVolumeControlSubscribe = “VolumeControl.Subscribe”
- kMediaControlAny = “Media.Control.Any”

All Capability header files also define a constant array of all capabilities defined in that header (ex. kVolumeControl-Capabilities).

AND/OR Filtering

CapabilityFilter is an AND filter. A ConnectableDevice would need to satisfy all conditions of a CapabilityFilter to pass.

[DiscoveryManager capabilityFilters] is an OR filter. a ConnectableDevice only needs to satisfy one condition (CapabilityFilter) to pass.

Examples

Filter for all devices that support video playback AND any media controls AND volume up/down.

```
NSArray *capabilities = @[
    kMediaPlayerPlayVideo,
    kMediaControlAny,
    kVolumeControlVolumeUpDown
];

CapabilityFilter *filter =
    [CapabilityFilter filterWithCapabilities:capabilities];

[[DiscoveryManager sharedManager] setCapabilityFilters:@[filter]];
```

Filter for all devices that support (video playback AND any media controls AND volume up/down) OR (image display).

```
NSArray *videoCapabilities = @[
    kMediaPlayerPlayVideo,
    kMediaControlAny,
    kVolumeControlVolumeUpDown
];

NSArray *imageCapabilities = @[
    kMediaPlayerDisplayImage
];

CapabilityFilter *videoFilter =
    [CapabilityFilter filterWithCapabilities:videoCapabilities];
CapabilityFilter *imageFilter =
    [CapabilityFilter filterWithCapabilities:imageCapabilities];

[[DiscoveryManager sharedManager] setCapabilityFilters:@[videoFilter, imageFilter]];
```

Properties

NSArray * capabilities Array of capabilities required by this filter. This property is readonly use the `addCapability` or `addCapabilities` to build this object.

Methods

+ (**CapabilityFilter ***) **filterWithCapabilities:(NSArray *)capabilities** Create a `CapabilityFilter` with the given array required capabilities.

Parameters

- capabilities – Capabilities to be added to the new filter

- (**void**) **addCapability:(NSString *)capability** Add a required capability to the filter.

Parameters

- capability – Capability name to add (see capability header files for NSString constants)

- (**void**) **addCapabilities:(NSArray *)capabilities** Add array of required capabilities to the filter.

Parameters

- capabilities – List of capability names (see capability header files for NSString constants)

DevicePicker

Overview

The `DevicePicker` is provided by the `DiscoveryManager` as a simple way for you to present a list of available devices to your users.

In Depth

The `DevicePicker` takes a sender parameter on the `showPicker` method. The sender parameter is used to display a popover from a particular `UIView` on iPads.

You should not attempt to instantiate the DevicePicker on your own. Instead, get the reference from the DeviceManager with `[[DeviceManager sharedManager] devicePicker];`

Properties

id<DevicePickerDelegate> delegate Delegate that receives selected/cancelled messages.

BOOL shouldAnimatePicker When the showPicker method is called, it can animate onto the screen if this value is set to YES. This value will also be used to determine if the picker should animate when it is dismissed.

BOOL shouldAutoRotate When the device is rotated, the DevicePicker can automatically adjust the view to compensate. Default is NO.

ConnectableDevice * currentDevice If you wish to show a checkmark next to a device in the picker, you can set that device object to currentDevice. The setter for currentDevice will also reload the tableView in the picker.

Methods

- **(void) showPicker:(id)sender** This method will animate the picker onto the screen. On iPad, the picker will appear as a popover view and will animate from the sender object, if you provide one. On iPhone, the picker will appear as a full-screen table view that will animate up from the bottom of the screen. This picker will animate in real time with additions, losses, and updates of ConnectableDevices.

Parameters:

- sender – On iPad, this should be a UIView for the popover view to animate from. On iPhone, this property is ignored.

- **(void) showActionSheet:(id)sender** This method will animate an action sheet onto the screen containing a button for each discovered ConnectableDevice. Due to the nature of action sheets, it is not possible to update the action sheet after it has appeared. It is recommended to use the showPicker: method if you want a picker that will update in real time.

Parameters:

- sender – The UIView that the action sheet should appear in

DevicePickerDelegate

The DevicePickerDelegate will receive a message when the user cancels or selects a ConnectableDevice from the DevicePicker list. This is the preferred method of selecting a device from DiscoveryManager.

Methods

- **(void) devicePicker:(DevicePicker *)picker didSelectDevice:(ConnectableDevice *)device** When the user selects a ConnectableDevice from the DevicePicker's list, this method will be called with the selected ConnectableDevice.

Parameters:

- picker – DevicePicker that device was selected from
- **didSelectDevice:** device – ConnectableDevice that was selected by the user

- (void) **devicePicker:(DevicePicker *)picker didCancelWithError:(NSError *)error** This method is called if the user presses the cancel button in the picker or if Connect SDK forces a cancellation. If Connect SDK forces a cancellation, there will be an NSError object passed with the reason.

Parameters:

- picker – DevicePicker that was cancelled
- **didCancelWithError:** error – NSError with a description of the failure

DiscoveryManager

Overview

At the heart of Connect SDK is DiscoveryManager, a multi-protocol service discovery engine with a pluggable architecture. Much of your initial experience with Connect SDK will be with the DiscoveryManager class, as it consolidates discovered service information into ConnectableDevice objects.

In depth

DiscoveryManager supports discovering services of differing protocols by using DiscoveryProviders. Many services are discoverable over [SSDP](#) and are registered to be discovered with the SSDPDiscoveryProvider class.

As services are discovered on the network, the DiscoveryProviders will notify DiscoveryManager. DiscoveryManager is capable of attributing multiple services, if applicable, to a single ConnectableDevice instance. Thus, it is possible to have a mixed-mode ConnectableDevice object that is theoretically capable of more functionality than a single service can provide.

DiscoveryManager keeps a running list of all discovered devices and maintains a filtered list of devices that have satisfied any of your CapabilityFilters. This filtered list is used by the DevicePicker when presenting the user with a list of devices.

Only one instance of the DiscoveryManager should be in memory at a time. To assist with this, DiscoveryManager has singleton accessors at `sharedManager` and `sharedManagerWithDeviceStore:`.

Example:

```
DiscoveryManager *discoveryManager = [DiscoveryManager sharedManager];
discoveryManager.delegate = self; // set delegate to listen for discovery events
[discoveryManager startDiscovery];
```

Properties

id<DiscoveryManagerDelegate> delegate Delegate which should receive discovery updates. It is not necessary to set this delegate property unless you are implementing your own device picker. Connect SDK provides a default DevicePicker which acts as a DiscoveryManagerDelegate, and should work for most cases.

If you have provided a `capabilityFilters` array, the delegate will only receive update messages for ConnectableDevices which satisfy at least one of the CapabilityFilters. If no `capabilityFilters` array is provided, the delegate will receive update messages for all ConnectableDevice objects that are discovered.

NSArray * capabilityFilters A ConnectableDevice will be displayed in the DevicePicker and `compatibleDevices` array if it matches any of the CapabilityFilter objects in this array.

***DeviceServicePairingLevel* pairingLevel** The pairingLevel property determines whether capabilities that require pairing (such as entering a PIN) will be available.

If pairingLevel is set to DeviceServicePairingLevelOn, ConnectableDevices that require pairing will prompt the user to pair when connecting to the ConnectableDevice.

If pairingLevel is set to DeviceServicePairingLevelOff (the default), connecting to the device will avoid requiring pairing if possible but some capabilities may not be available.

id<ConnectableDeviceStore> deviceStore ConnectableDeviceStore object which loads & stores references to all discovered devices. Pairing codes/keys, SSL certificates, recent access times, etc are kept in the device store.

ConnectableDeviceStore is a protocol which may be implemented as needed. A default implementation, DefaultConnectableDeviceStore, exists for convenience and will be used if no other device store is provided.

In order to satisfy user privacy concerns, you should provide a UI element in your app which exposes the ConnectableDeviceStore removeAll method.

To disable the ConnectableDeviceStore capabilities of Connect SDK, set this value to nil. This may be done at the time of instantiation with [DiscoveryManager sharedManagerWithDeviceStore:nil].

BOOL useDeviceStore Whether pairing state will be automatically loaded/saved in the deviceStore. This property is not available for direct modification. To disable the device store,

Methods

+ **(instancetype) sharedManager** Singleton accessor for DiscoveryManager. This method calls sharedManagerWithDeviceStore: and passes an instance of DefaultConnectableDeviceStore.

+ **(instancetype) sharedManagerWithDeviceStore:(id<ConnectableDeviceStore>)deviceStore** Singleton accessor for DiscoveryManager, will initialize singleton with reference to a custom ConnectableDeviceStore object.

Parameters:

- deviceStore – (optional) An object which implements the ConnectableDeviceStore protocol to be used for save/load of device information. You may provide nil to completely disable the device store capabilities of the SDK.
- **(NSDictionary *) compatibleDevices** Filtered list of discovered ConnectableDevices, limited to devices that match at least one of the CapabilityFilters in the capabilityFilters array. Each ConnectableDevice object is keyed against its current IP address.
- **(NSDictionary *) allDevices** List of all devices discovered by DiscoveryManager. Each ConnectableDevice object is keyed against its current IP address.
- **(void) startDiscovery** Start scanning for devices on the local network.
- **(void) stopDiscovery** Stop scanning for devices.

This method will be called when your app enters a background state. When your app resumes, startDiscovery will be called.

- **(DevicePicker *) devicePicker** Get a DevicePicker to show compatible ConnectableDevices that have been found by DiscoveryManager.

Returns: DevicePickerDevicePicker singleton for use in picking devices

DiscoveryManagerDelegate

Overview

The `DiscoveryManagerDelegate` will receive events on the addition/removal/update of `ConnectableDevice` objects.

In Depth

It is important to note that, unless you are implementing your own device picker, this delegate is not needed in your code. Connect SDK's `DevicePicker` internally acts a separate delegate to the `DiscoveryManager` and handles all of the same method calls.

Methods

- **(void) discoveryManager:(*DiscoveryManager **)manager didFindDevice:(*ConnectableDevice **)device** This method will be fired upon the first discovery of one of a `ConnectableDevice`'s `DeviceServices`.

Parameters:

- `manager` – `DiscoveryManager` that found device
- **didFindDevice:** `device` – `ConnectableDevice` that was found

- **(void) discoveryManager:(*DiscoveryManager **)manager didLoseDevice:(*ConnectableDevice **)device** This method is called when connections to all of a `ConnectableDevice`'s `DeviceServices` are lost. This will usually happen when a device is powered off or loses internet connectivity.

Parameters:

- `manager` – `DiscoveryManager` that lost device
- **didLoseDevice:** `device` – `ConnectableDevice` that was lost

- **(void) discoveryManager:(*DiscoveryManager **)manager didUpdateDevice:(*ConnectableDevice **)device** This method is called when a `ConnectableDevice` gains or loses a `DeviceService` in discovery.

Parameters:

- `manager` – `DiscoveryManager` that updated device
- **didUpdateDevice:** `device` – `ConnectableDevice` that was updated

- **(void) discoveryManager:(*DiscoveryManager **)manager didFailWithError:(*NSError **)error** In the event of an error in the discovery phase, this method will be called.

Parameters:

- `manager` – `DiscoveryManager` that experienced the error
- **didFailWithError:** `error` – `NSError` with a description of the failure

5.16.2 Device

ConnectableDevice

Overview

`ConnectableDevice` serves as a normalization layer between your app and each of the device's services. It consolidates a lot of key data about the physical device and provides access to underlying functionality.

In Depth

ConnectableDevice consolidates some key information about the physical device, including model name, friendly name, ip address, connected DeviceService names, etc. In some cases, it is not possible to accurately select which DeviceService has the best friendly name, model name, etc. In these cases, the values of these properties are dependent upon the order of DeviceService discovery.

To be informed of any ready/pairing/disconnect messages from each of the DeviceService, you must set a delegate.

ConnectableDevice exposes capabilities that exist in the underlying DeviceServices such as TV Control, Media Player, Media Control, Volume Control, etc. These capabilities, when accessed through the ConnectableDevice, will be automatically chosen from the most suitable DeviceService by using that DeviceService's CapabilityPriorityLevel.

Properties

id<*ConnectableDeviceDelegate* > **delegate** Delegate which should receive messages on certain events.

NSString * id Universally unique ID of this particular ConnectableDevice object, persists between sessions in ConnectableDeviceStore for connected devices

NSString * address Current IP address of the ConnectableDevice.

NSString * friendlyName An estimate of the ConnectableDevice's current friendly name.

NSString * modelName An estimate of the ConnectableDevice's current model name.

NSString * modelNumber An estimate of the ConnectableDevice's current model number.

NSString * lastKnownIPAddress Last IP address this ConnectableDevice was discovered at.

NSString * lastSeenOnWifi Name of the last wireless network this ConnectableDevice was discovered on.

double lastConnected Last time (in seconds from 1970) that this ConnectableDevice was connected to.

double lastDetection Last time (in seconds from 1970) that this ConnectableDevice was detected.

BOOL isConnectable Whether the device has any DeviceServices that require an active connection (websocket, HTTP registration, etc)

BOOL connected Whether all the DeviceServices are connected.

NSArray * services Array of all currently discovered DeviceServices this ConnectableDevice has associated with it.

BOOL hasServices Whether the ConnectableDevice has any running DeviceServices associated with it.

NSArray * capabilities A combined list of all capabilities that are supported among the detected DeviceServices.

Methods

- **(void) connect** Enumerates through all DeviceServices and attempts to connect to each of them. When all of a ConnectableDevice's DeviceServices are ready to receive commands, the ConnectableDevice will send a connectableDeviceReady: message to its delegate.

It is always necessary to call connect on a ConnectableDevice, even if it contains no connectable DeviceServices.

- **(void) disconnect** Enumerates through all DeviceServices and attempts to disconnect from each of them.

- **(void) addService:(DeviceService *)service** Adds a DeviceService to the ConnectableDevice instance. Only one instance of each DeviceService type (webOS, Netcast, etc) may be attached to a single ConnectableDevice instance. If a device contains your service type already, your service will not be added.

Parameters:

- service – DeviceService to be added to the ConnectableDevice

- **(void) removeServiceWithId:(NSString *)serviceId** Removes a DeviceService from the ConnectableDevice instance. serviceId is used as the identifier because only one instance of each DeviceService type may be attached to a single ConnectableDevice instance.

Parameters:

- serviceId – Id of the DeviceService to be removed from the ConnectableDevice

- **(DeviceService *) serviceWithName:(NSString *)serviceId** Obtains a service from the device with the provided serviceId

Parameters:

- serviceId – Service ID of the targeted DeviceService (webOS, Netcast, DLNA, etc)

Returns: DeviceService with the specified serviceId or nil, if none exists

- **(BOOL) hasCapability:(NSString *)capability** Test to see if the capabilities array contains a given capability. See the individual Capability classes for acceptable capability values.

It is possible to append a wildcard search term `.Any` to the end of the search term. This method will return true for capabilities that match the term up to the wildcard.

Example: `Launcher.App.Any`

Parameters:

- capability – Capability to test against

- **(BOOL) hasCapabilities:(NSArray *)capabilities** Test to see if the capabilities array contains a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability:` for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – Array of capabilities to test against

- **(BOOL) hasAnyCapability:(NSArray *)capabilities** Test to see if the capabilities array contains at least one capability in a given set of capabilities. See the individual Capability classes for acceptable capability values.

See `hasCapability:` for a description of the wildcard feature provided by this method.

Parameters:

- capabilities – Array of capabilities to test against

- **(void) setPairingType:(DeviceServicePairingType)pairingType** Set the type of pairing for the ConnectableDevice services. By default the value will be `DeviceServicePairingTypeNone`

For `WebOSTV`'s If `pairingType` is set to `DeviceServicePairingTypeFirstScreen`(default), the device will prompt the user to pair when connecting to the `ConnectableDevice`.

If `pairingType` is set to `DeviceServicePairingTypePinCode`, the device will prompt the user to enter a pin to pair when connecting to the `ConnectableDevice`.

Parameters:

- pairingType – value to be set for the device service from `DeviceServicePairingType`

- (id<Launcher >) **launcher**

- (id<ExternalInputControl >) **externalInputControl** Accessor for highest priority Launcher object

- (id<MediaPlayer >) **mediaPlayer** Accessor for highest priority ExternalInputControl object

- (**id**<*MediaControl*>) **mediaControl** Accessor for highest priority MediaPlayer object
- (**id**<*VolumeControl*>) **volumeControl** Accessor for highest priority MediaControl object
- (**id**<*TVControl*>) **tvControl** Accessor for highest priority VolumeControl object
- (**id**<*KeyControl*>) **keyControl** Accessor for highest priority TVControl object
- (**id**<*TextInputControl*>) **textInputControl** Accessor for highest priority KeyControl object
- (**id**<*MouseControl*>) **mouseControl** Accessor for highest priority TextInputControl object
- (**id**<*PowerControl*>) **powerControl** Accessor for highest priority MouseControl object
- (**id**<*ToastControl*>) **toastControl** Accessor for highest priority PowerControl object
- (**id**<*WebAppLauncher*>) **webAppLauncher** Accessor for highest priority ToastControl object

ConnectableDeviceDelegate

ConnectableDeviceDelegate allows for a class to receive messages about ConnectableDevice connection, disconnect, and update events.

It also serves as a delegate proxy for message handling when connecting and pairing with each of a ConnectableDevice's DeviceServices. Each of the DeviceService proxy methods are optional and would only be useful in a few use cases.

- providing your own UI for the pairing process.
- **interacting directly and exclusively with a single type of** DeviceService

Methods

- (**void**) **connectableDeviceReady:(ConnectableDevice *)device** A ConnectableDevice sends out a ready message when all of its connectable DeviceServices have been connected and are ready to receive commands.

Parameters:

- device – ConnectableDevice that is ready for commands.

- (**void**) **connectableDeviceDisconnected:(ConnectableDevice *)device withError:(NSError *)error** When all of a ConnectableDevice's DeviceServices have become disconnected, the disconnected message is sent.

Parameters:

- device – ConnectableDevice that has been disconnected.
- **withError:** error

- (**void**) **connectableDevice:(ConnectableDevice *)device capabilitiesAdded:(NSArray *)added removed:(NSArray *)removed** When a ConnectableDevice finds & loses DeviceServices, that ConnectableDevice will experience a change in its collective capabilities list. When such a change occurs, this message will be sent with arrays of capabilities that were added & removed.

This message will allow you to decide when to stop/start interacting with a ConnectableDevice, based off of its supported capabilities.

Parameters:

- device – ConnectableDevice that has experienced a change in capabilities
- **capabilitiesAdded:** added – NSArray of capabilities that are new to the ConnectableDevice
- **removed:** removed – NSArray of capabilities that the ConnectableDevice has lost

- (void) **connectableDevice:(ConnectableDevice *)device connectionFailedWithError:(NSError *)error** This method is called when the connection to the ConnectableDevice has failed.

Parameters:

- **device** – ConnectableDevice that has failed to connect
- **connectionFailedWithError:** error – NSError with a description of the failure

- (void) **connectableDeviceConnectionRequired:(ConnectableDevice *)device forService:(DeviceService *)service** DeviceService delegate proxy method.

This method is called when a DeviceService requires an active connection. This will be the case for DeviceServices that send messages over websockets (webOS, etc) and DeviceServices that require pairing to send messages (Netcast, etc).

Parameters:

- **device** – ConnectableDevice containing the DeviceService
- **forService:** service – DeviceService which requires a connection

- (void) **connectableDeviceConnectionSuccess:(ConnectableDevice *)device forService:(DeviceService *)service** DeviceService delegate proxy method.

This method is called when a DeviceService has successfully connected.

Parameters:

- **device** – ConnectableDevice containing the DeviceService
- **forService:** service – DeviceService which has connected

- (void) **connectableDevice:(ConnectableDevice *)device service:(DeviceService *)service disconnectedWithError:(NSError *)error** DeviceService delegate proxy method.

This method is called when a DeviceService becomes disconnected.

Parameters:

- **device** – ConnectableDevice containing the DeviceService
- **service:** service – DeviceService which has disconnected
- **disconnectedWithError:** error – NSError with a description of any errors causing the disconnect. If this value is nil, then the disconnect was clean/expected.

- (void) **connectableDevice:(ConnectableDevice *)device service:(DeviceService *)service didFailConnectWithError:(NSError *)error** DeviceService delegate proxy method.

This method is called when a DeviceService fails to connect.

Parameters:

- **device** – ConnectableDevice containing the DeviceService
- **service:** service – DeviceService which has failed to connect
- **didFailConnectWithError:** error – NSError with a description of the failure

- (void) **connectableDevice:(ConnectableDevice *)device service:(DeviceService *)service pairingRequiredOfType:(int)pairingType** DeviceService delegate proxy method.

This method is called when a DeviceService tries to connect and finds out that it requires pairing information from the user.

Parameters:

- **device** – ConnectableDevice containing the DeviceService
- **service**: service – DeviceService that requires pairing
- **pairingRequiredOfType**: pairingType – DeviceServicePairingType that the DeviceService requires
- **withData**: pairingData – Any data that might be required for the pairing process, will usually be nil

- (void) **connectableDevicePairingSuccess:(ConnectableDevice *)device service:(DeviceService *)service**
DeviceService delegate proxy method.

This method is called when a DeviceService completes the pairing process.

Parameters:

- **device** – ConnectableDevice containing the DeviceService
- **service**: service – DeviceService that has successfully completed pairing

- (void) **connectableDevice:(ConnectableDevice *)device service:(DeviceService *)service pairingFailedWithError:(NSError *)error**
DeviceService delegate proxy method.

This method is called when a DeviceService fails to complete the pairing process.

Parameters:

- **device** – ConnectableDevice containing the DeviceService
- **service**: service – DeviceService that has failed to complete pairing
- **pairingFailedWithError**: error – NSError with a description of the failure

ServiceCommand

Properties

id<ServiceCommandDelegate> delegate

SuccessBlock callbackComplete

FailureBlock callbackError

NSString * HTTPMethod

id payload

NSURL * target

Methods

- (instancetype) **initWithDelegate:(id<ServiceCommandDelegate>)delegate target:(NSURL *)url payload:(id)payload**

Parameters:

- **delegate**
- **target**: url
- **payload**: payload

- (void) **send**

+ (instancetype) **commandWithDelegate:(id<ServiceCommandDelegate>)delegate target:(NSURL *)url payload:(id)payload**

Parameters:

- delegate
- **target:** url
- **payload:** payload

ServiceSubscription

extends ServiceCommand

Properties

int callId

NSMutableArray * successCalls

NSMutableArray * failureCalls

BOOL isSubscribed

Methods

- (instancetype) initWithDelegate:(id<ServiceCommandDelegate>)delegate target:(NSURL *)target payload:(id)payload callId:(id)callId

Parameters:

- delegate
- **target:** target
- **payload:** payload
- **callId:** callId

- (void) addSuccess:(id)success **Parameters:**

- success – Optional id to be called on success

- (void) addFailure:(FailureBlock)failure **Parameters:**

- failure – Optional FailureBlock to be called on failure

- (void) subscribe

- (void) unsubscribe

+(instancetype) subscriptionWithDelegate:(id<ServiceCommandDelegate>)delegate target:(NSURL *)url payload:(id)payload callId:(id)callId

Parameters:

- delegate
- **target:** url
- **payload:** payload
- **callId:** callId

Inherited Methods

- (instancetype) initWithDelegate:(id<ServiceCommandDelegate>)delegate target:(NSURL *)url payload:(id)payload

Parameters:

- delegate
- **target:** url
- **payload:** payload

- (void) send

• (instancetype) commandWithDelegate:(id<ServiceCommandDelegate>)delegate target:(NSURL *)url payload:(id)payload

Parameters:

- delegate
- **target:** url
- **payload:** payload

5.16.3 Device Services

AirPlayService

extends DeviceService

AirPlayService provides media playback/control & web app launching (iOS only) capabilities for Apple TV devices.

AirPlay-enabled speakers are not currently supported by Connect SDK.

Default functionality

Out of the box, AirPlayService will only support web app launching through AirPlay mirroring. AirPlayService also provides a Media mode, in which HTTP commands will be sent to the AirPlay device to play and control media files (image, video, audio). Due to certain limitations of the AirPlay protocol, you may only support web apps OR media capabilities through Connect SDK. You may still directly access AirPlay APIs through AVPlayer, MPMoviePlayerController, UIWebView, audio routing, etc.

To set the capability mode for the AirPlayService, see the `setAirPlayServiceMode:` static method on the AirPlayService class.

Methods

+ (*AirPlayServiceMode*) **serviceMode** Returns the current AirPlayServiceMode

+ (void) **setAirPlayServiceMode:(AirPlayServiceMode)serviceMode** Sets the AirPlayService mode. This property should be set before DiscoveryManager is set for the first time.

Parameters:

- serviceMode

Inherited Methods

+ (NSDictionary *) **discoveryParameters** A dictionary of keys/values that will be used by the DiscoveryProvider used to discover this DeviceService. Some keys that are used are: service name, SSDP filter, etc.

+ (*DeviceService* *) **deviceServiceWithClass:(Class)_class serviceConfig:(ServiceConfig *)serviceConfig**
Parameters:

- `_class`
- **serviceConfig:** serviceConfig

+ (BOOL) **shouldDisconnectOnBackground** Static property that determines whether a DeviceService subclass should shut down communication channels when the app enters a background state. This may be helpful for apps that need to communicate with web apps from the background. This property may not be applicable to all DeviceService subclasses.

Sets the shouldDisconnectOnBackground static property. This property should be set before starting Discovery-Manager for the first time.

+ (void) **setShouldDisconnectOnBackground:(BOOL)shouldDisconnectOnBackground** **Parameters:**

- shouldDisconnectOnBackground

- (instancetype) **initWithServiceConfig:(ServiceConfig *)serviceConfig** **Parameters:**

- serviceConfig

- (BOOL) **hasCapability:(NSString *)capability** **Parameters:**

- capability

- (BOOL) **hasCapabilities:(NSArray *)capabilities** **Parameters:**

- capabilities

- (BOOL) **hasAnyCapability:(NSArray *)capabilities** **Parameters:**

- capabilities

- (void) **connect** Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate. If the connection attempt reveals that pairing is required, the DeviceServiceDelegate will also be notified in that event.

- (void) **disconnect** Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate.

- (void) **pairWithData:(id)pairingData** Will attempt to pair with the DeviceService with the provided pairingData. The failure/success will be reported back to the DeviceServiceDelegate.

Parameters:

- pairingData – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

- (void) **closeLaunchSession:(LaunchSession *)launchSession success:(SuccessBlock)success failure:(FailureBlock)failure**

Every LaunchSession object has an associated DeviceService. Internally, LaunchSession's close method proxies to it's DeviceService's closeLaunchSession method. If, for some reason, your LaunchSession loses it's DeviceService reference, you can call this closeLaunchSession method directly.

Parameters:

- launchSession – LaunchSession to be closed
- **success:** success – (optional) SuccessBlock to be called on success

- **failure:** failure – (optional) FailureBlock to be called on failure

AirPlayServiceHTTPKeepAlive

The class is responsible for maintaining an AirPlay connection alive by sending periodic requests.

Properties

CGFloat interval The interval between keep-alive requests, in seconds. 50 by default.

id<ServiceCommandDelegate> commandDelegate An object that sends AirPlay commands.

NSURL * commandURL The base URL for commands.

Methods

- (instancetype) **initWithInterval:(CGFloat)interval andCommandDelegate:(id<ServiceCommandDelegate>)commandDelegate** Designated initializer, setting the interval and command delegate.

Parameters:

- interval
- **andCommandDelegate:** commandDelegate

- (instancetype) **initWithCommandDelegate:(id<ServiceCommandDelegate>)commandDelegate** Initializer that sets the command delegate.

Parameters:

- commandDelegate

- (void) **startTimer** Schedules sending keep-alive requests. The first one will be sent after the specified `interval`.

- (void) **stopTimer** Stops sending keep-alive requests.

AirPlayServiceMode

The values in this enum type define what capabilities should be supported by the AirPlayService.

Properties

AirPlayServiceModeWebApp Enables support for web apps via Apple's [External Display APIs](#)

AirPlayServiceModeMedia Enables support for media (image, video, audio) by way of [HTTP commands](#)

CastService

extends DeviceService

CastService provides capabilities for Google Chromecast devices. CastService acts as a layer on top of Google's own Cast SDK, and requires the Cast SDK library to function. CastService provides the following functionality:

- Media playback
- Media control

- Web app launching & two-way communication
- Volume control

Using Connect SDK for discovery/control of Chromecast devices will result in your app complying with the Google Cast SDK [terms of service](#).

To learn more about Cast SDK, visit the [Google Cast SDK Developer site](#).

Properties

GCKDeviceManager * castDeviceManager The GCKDeviceManager that CastService is using internally to manage devices.

GCKDevice * castDevice The GCKDevice object that CastService is using internally for device information.

CastServiceChannel * castServiceChannel The CastServiceChannel is used for app-to-app communication that is handling by the Connect SDK JavaScript Bridge.

GCKMediaControlChannel * castMediaControlChannel The GCKMediaControlChannel that the CastService is using to send media events to the connected web app.

NSString * castWebAppId The CastService will launch the specified web app id.

Inherited Methods

+ (NSDictionary *) **discoveryParameters** A dictionary of keys/values that will be used by the DiscoveryProvider used to discover this DeviceService. Some keys that are used are: service name, SSDP filter, etc.

+ (*DeviceService* *) **deviceServiceWithClass:(Class)_class serviceConfig:(ServiceConfig *)serviceConfig**
Parameters:

- *_class*
- **serviceConfig:** serviceConfig

+ (BOOL) **shouldDisconnectOnBackground** Static property that determines whether a DeviceService subclass should shut down communication channels when the app enters a background state. This may be helpful for apps that need to communicate with web apps from the background. This property may not be applicable to all DeviceService subclasses.

Sets the shouldDisconnectOnBackground static property. This property should be set before starting DiscoveryManager for the first time.

+ (void) **setShouldDisconnectOnBackground:(BOOL)shouldDisconnectOnBackground** **Parameters:**

- shouldDisconnectOnBackground

- (instancetype) **initWithServiceConfig:(ServiceConfig *)serviceConfig** **Parameters:**

- serviceConfig

- (BOOL) **hasCapability:(NSString *)capability** **Parameters:**

- capability

- (BOOL) **hasCapabilities:(NSArray *)capabilities** **Parameters:**

- capabilities

- (BOOL) **hasAnyCapability:(NSArray *)capabilities** **Parameters:**

- capabilities

- **(void) connect** Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate. If the connection attempt reveals that pairing is required, the DeviceServiceDelegate will also be notified in that event.
- **(void) disconnect** Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate.
- **(void) pairWithData:(id)pairingData** Will attempt to pair with the DeviceService with the provided pairingData. The failure/success will be reported back to the DeviceServiceDelegate.

Parameters:

- pairingData –
Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

- **(void) closeLaunchSession:(LaunchSession *)launchSession success:(SuccessBlock)success failure:(FailureBlock)failure**
Every LaunchSession object has an associated DeviceService. Internally, LaunchSession’s close method proxies to it’s DeviceService’s closeLaunchSession method. If, for some reason, your LaunchSession loses it’s DeviceService reference, you can call this closeLaunchSession method directly.

Parameters:

- launchSession – LaunchSession to be closed
- **success:** success – (optional) SuccessBlock to be called on success
- **failure:** failure – (optional) FailureBlock to be called on failure

DIALService

extends DeviceService

DIALService is a full implementation of the Discover And Launch (DIAL) protocol specification. DIALService is used to launch & close apps on DIAL-enabled devices. It can also be used to probe for an app’s existence on a DIAL-enabled device. DIAL commands occur over HTTP.

See the [DIAL protocol specification](#) for more information.

Methods

- + **(void) registerApp:(NSString *)appId** Registers an app ID to be checked upon discovery of this device. If the app is found on the target device, the DIALService will gain the “Launcher.” capability, where is the value of the appId parameter.

This method must be called before starting DiscoveryManager for the first time.

Parameters:

- appId – ID of the app to be checked for

Inherited Methods

- + **(NSDictionary *) discoveryParameters** A dictionary of keys/values that will be used by the DiscoveryProvider used to discover this DeviceService. Some keys that are used are: service name, SSDP filter, etc.
- + **(DeviceService *) deviceServiceWithClass:(Class)_class serviceConfig:(ServiceConfig *)serviceConfig**

Parameters:

- `_class`
 - **serviceConfig**: serviceConfig
- + (BOOL) **shouldDisconnectOnBackground** Static property that determines whether a DeviceService subclass should shut down communication channels when the app enters a background state. This may be helpful for apps that need to communicate with web apps from the background. This property may not be applicable to all DeviceService subclasses.
- Sets the shouldDisconnectOnBackground static property. This property should be set before starting Discovery-Manager for the first time.
- + (void) **setShouldDisconnectOnBackground:(BOOL)shouldDisconnectOnBackground** Parameters:
- shouldDisconnectOnBackground
- (instancetype) **initWithServiceConfig:(ServiceConfig *)serviceConfig** Parameters:
- serviceConfig
- (BOOL) **hasCapability:(NSString *)capability** Parameters:
- capability
- (BOOL) **hasCapabilities:(NSArray *)capabilities** Parameters:
- capabilities
- (BOOL) **hasAnyCapability:(NSArray *)capabilities** Parameters:
- capabilities
- (void) **connect** Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate. If the connection attempt reveals that pairing is required, the DeviceServiceDelegate will also be notified in that event.
- (void) **disconnect** Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate.
- (void) **pairWithData:(id)pairingData** Will attempt to pair with the DeviceService with the provided pairingData. The failure/success will be reported back to the DeviceServiceDelegate.

Parameters:

- pairingData – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).
- (void) **closeLaunchSession:(LaunchSession *)launchSession success:(SuccessBlock)success failure:(FailureBlock)failure**
Every LaunchSession object has an associated DeviceService. Internally, LaunchSession's close method proxies to its DeviceService's closeLaunchSession method. If, for some reason, your LaunchSession loses its DeviceService reference, you can call this closeLaunchSession method directly.

Parameters:

- launchSession – LaunchSession to be closed
- **success**: success – (optional) SuccessBlock to be called on success
- **failure**: failure – (optional) FailureBlock to be called on failure

DLNAService

extends DeviceService

DLNASService is a rough control implementation for the UPnP AVTransport, MediaRenderer, and RenderingControl services. DLNA commands & events occur over HTTP.

This service currently exists for the sole purpose of providing media control/playback functionality for the Net-castTVService. DiscoveryManager is currently set up to ignore any DLNA devices that are not manufactured by LG. It is not recommended to remove this restriction, as the DLNASService implementation is not complete.

To learn more about the protocols in use by DLNASService, check out the following documents.

- [UPnP](#)
- [AVTransport Service](#)
- [MediaRenderer Device](#)
- [RenderingControl Service](#)

Inherited Methods

+ (NSDictionary *) **discoveryParameters** A dictionary of keys/values that will be used by the DiscoveryProvider used to discover this DeviceService. Some keys that are used are: service name, SSDP filter, etc.

+ (*DeviceService* *) **deviceServiceWithClass:(Class)_class serviceConfig:(ServiceConfig *)serviceConfig**
Parameters:

- `_class`
- **serviceConfig:** serviceConfig

+ (BOOL) **shouldDisconnectOnBackground** Static property that determines whether a DeviceService subclass should shut down communication channels when the app enters a background state. This may be helpful for apps that need to communicate with web apps from the background. This property may not be applicable to all DeviceService subclasses.

Sets the shouldDisconnectOnBackground static property. This property should be set before starting DiscoveryManager for the first time.

+ (void) **setShouldDisconnectOnBackground:(BOOL)shouldDisconnectOnBackground** Parameters:

- shouldDisconnectOnBackground

- (instancetype) **initWithServiceConfig:(ServiceConfig *)serviceConfig** Parameters:

- serviceConfig

- (BOOL) **hasCapability:(NSString *)capability** Parameters:

- capability

- (BOOL) **hasCapabilities:(NSArray *)capabilities** Parameters:

- capabilities

- (BOOL) **hasAnyCapability:(NSArray *)capabilities** Parameters:

- capabilities

- (void) **connect** Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate. If the connection attempt reveals that pairing is required, the DeviceServiceDelegate will also be notified in that event.

- (void) **disconnect** Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate.

- **(void) pairWithData:(id)pairingData** Will attempt to pair with the DeviceService with the provided pairingData. The failure/success will be reported back to the DeviceServiceDelegate.

Parameters:

- pairingData – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

- **(void) closeLaunchSession:(LaunchSession *)launchSession success:(SuccessBlock)success failure:(FailureBlock)failure**

Every LaunchSession object has an associated DeviceService. Internally, LaunchSession’s close method proxies to it’s DeviceService’s closeLaunchSession method. If, for some reason, your LaunchSession loses it’s DeviceService reference, you can call this closeLaunchSession method directly.

Parameters:

- launchSession – LaunchSession to be closed
- **success:** success – (optional) SuccessBlock to be called on success
- **failure:** failure – (optional) FailureBlock to be called on failure

DeviceService

Overview

From a high-level perspective, DeviceService completely abstracts the functionality of a particular service/protocol (webOS TV, Netcast TV, Chromecast, Roku, DIAL, etc).

In Depth

DeviceService is an abstract class that is meant to be extended. You shouldn’t ever use DeviceService directly, unless extending it to provide support for an additional service/protocol.

Immediately after discovery of a DeviceService, DiscoveryManager will set the DeviceService’s delegate to the ConnectableDevice that owns the DeviceService. You should not change the delegate unless you intend to manage the lifecycle of that service. The DeviceService will proxy all of its delegate method calls through the ConnectableDevice’s ConnectableDeviceDelegate.

Connection & Pairing

Your ConnectableDevice object will let you know if you need to connect or pair to any services.

Capabilities

All DeviceService objects have a group of capabilities. These capabilities can be implemented by any object, and that object will be returned when you call the DeviceService’s capability methods (launcher, mediaPlayer, volumeControl, etc).

Properties

id<DeviceServiceDelegate> delegate Delegate object to receive DeviceService status messages. See note in the “In Depth” section about changing the DeviceServiceDelegate.

ServiceDescription * serviceDescription Object containing the discovered information about this DeviceService

ServiceConfig * serviceConfig Object containing persistence data about this DeviceService (pairing info, SSL certificates, etc)

NSString * serviceName Name of the DeviceService (webOS, Chromecast, etc)

NSArray * capabilities An array of capabilities supported by the DeviceService. This array may change based off a number of factors.

- DiscoveryManager's pairingLevel value
- Connect SDK framework version
- First screen device OS version
- First screen device configuration (apps installed, settings, etc)
- Physical region

BOOL connected Whether the DeviceService is currently connected

BOOL isConnectedable Whether the DeviceService requires an active connection or registration process

BOOL requiresPairing Whether the DeviceService requires pairing or not.

DeviceServicePairingType pairingType Type of pairing that this DeviceService requires. May be unknown until you try to connect.

id pairingData May contain useful information regarding pairing (pairing key length, etc)

Methods

+ (NSDictionary *) **discoveryParameters** A dictionary of keys/values that will be used by the DiscoveryProvider used to discover this DeviceService. Some keys that are used are: service name, SSDP filter, etc.

+ (*DeviceService* *) **deviceServiceWithClass:(Class)_class serviceConfig:(ServiceConfig *)serviceConfig**
Parameters:

- `_class`
- **serviceConfig:** serviceConfig

+ (BOOL) **shouldDisconnectOnBackground** Static property that determines whether a DeviceService subclass should shut down communication channels when the app enters a background state. This may be helpful for apps that need to communicate with web apps from the background. This property may not be applicable to all DeviceService subclasses.

Sets the shouldDisconnectOnBackground static property. This property should be set before starting DiscoveryManager for the first time.

+ (void) **setShouldDisconnectOnBackground:(BOOL)shouldDisconnectOnBackground** **Parameters:**

- shouldDisconnectOnBackground

- (instancetype) **initWithServiceConfig:(ServiceConfig *)serviceConfig** **Parameters:**

- serviceConfig

- (BOOL) **hasCapability:(NSString *)capability** **Parameters:**

- capability

- (BOOL) **hasCapabilities:(NSArray *)capabilities** **Parameters:**

- capabilities

- (BOOL) **hasAnyCapability:(NSArray *)capabilities** **Parameters:**

- capabilities

- (void) **connect** Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate. If the connection attempt reveals that pairing is required, the DeviceServiceDelegate will also be notified in that event.

- (void) **disconnect** Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate.

- (void) **pairWithData:(id)pairingData** Will attempt to pair with the DeviceService with the provided pairingData. The failure/success will be reported back to the DeviceServiceDelegate.

Parameters:

- pairingData –

Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

- (void) **closeLaunchSession:(LaunchSession *)launchSession success:(SuccessBlock)success failure:(FailureBlock)failure**
Every LaunchSession object has an associated DeviceService. Internally, LaunchSession's close method proxies to its DeviceService's closeLaunchSession method. If, for some reason, your LaunchSession loses its DeviceService reference, you can call this closeLaunchSession method directly.

Parameters:

- launchSession – LaunchSession to be closed
- **success:** success – (optional) SuccessBlock to be called on success
- **failure:** failure – (optional) FailureBlock to be called on failure

DeviceServiceDelegate

DeviceServiceDelegate allows your app to respond to each step of the connection and pairing processes, if needed. By default, a DeviceService's ConnectableDevice is set as the delegate. Changing a DeviceService's delegate will break the normal operation of Connect SDK and is discouraged. ConnectableDeviceDelegate provides proxy methods for all of the methods listed here.

Methods

- (void) **deviceServiceConnectionRequired:(DeviceService *)service** If the DeviceService requires an active connection (websocket, pairing, etc) this method will be called.

Parameters:

- service – DeviceService that requires connection

- (void) **deviceServiceConnectionSuccess:(DeviceService *)service** After the connection has been successfully established, and after pairing (if applicable), this method will be called.

Parameters:

- service – DeviceService that was successfully connected

- (void) **deviceService:(DeviceService *)service capabilitiesAdded:(NSArray *)added removed:(NSArray *)removed**
There are situations in which a DeviceService will update the capabilities it supports and propagate these changes to the DeviceService. Such situations include:

- on discovery, DIALService will reach out to detect if certain apps are installed

- on discovery, certain DeviceServices need to reach out for & region information

For more information on this particular method, see ConnectableDeviceDelegate's connectableDevice:capabilitiesAdded:removed: method.

Parameters:

- service – DeviceService that has experienced a change in capabilities
- **capabilitiesAdded:** added – NSArray of capabilities that are new to the DeviceService
- **removed:** removed – NSArray of capabilities that the DeviceService has lost

- (void) **deviceService:(DeviceService *)service disconnectedWithError:(NSError *)error** This method will be called on any disconnection. If error is nil, then the connection was clean and likely triggered by the responsible DiscoveryProvider or by the user.

Parameters:

- service – DeviceService that disconnected
- **disconnectedWithError:** error – NSError with a description of any errors causing the disconnect. If this value is nil, then the disconnect was clean/expected.

- (void) **deviceService:(DeviceService *)service didFailConnectWithError:(NSError *)error** Will be called if the DeviceService fails to establish a connection.

Parameters:

- service – DeviceService which has failed to connect
- **didFailConnectWithError:** error – NSError with a description of the failure

- (void) **deviceService:(DeviceService *)service pairingRequiredOfType:(DeviceServicePairingType)pairingType withData:(id)pairingData** If the DeviceService requires pairing, valuable data will be passed to the delegate via this method.

Parameters:

- service – DeviceService that requires pairing
- **pairingRequiredOfType:** pairingType – DeviceServicePairingType that the DeviceService requires
- **withData:** pairingData – Any object/data that might be required for the pairing process, will usually be nil

- (void) **deviceServicePairingSuccess:(DeviceService *)service** **Parameters:**

- service

- (void) **deviceService:(DeviceService *)service pairingFailedWithError:(NSError *)error** If there is any error in pairing, this method will be called.

Parameters:

- service – DeviceService that has failed to complete pairing
- **pairingFailedWithError:** error – NSError with a description of the failure

DeviceServicePairingLevel

Enumerated value for determining how a DeviceService should handle pairing when attempting to connect.

Properties

DeviceServicePairingLevelOff DeviceServices will never try to pair with a device

DeviceServicePairingLevelOn DeviceServices will try to pair with a device, if needed

DeviceServicePairingType

Type of pairing that is required by a particular DeviceService. This type will be passed along with the DeviceServiceDelegate deviceService:pairingRequiredOfType:withData: message.

Properties

DeviceServicePairingTypeNone DeviceService does not require pairing

DeviceServicePairingTypeFirstScreen DeviceService requires user interaction on the first screen (ex. pairing alert)

DeviceServicePairingTypePinCode First screen is displaying a pairing pin code that can be sent through the DeviceService

DeviceServicePairingTypeMixed DeviceService can pair with multiple pairing types (ex. first screen OR pin)

DeviceServicePairingTypeAirPlayMirroring DeviceService requires AirPlay mirroring to be enabled to connect

DeviceServicePairingTypeUnknown DeviceService pairing type is unknown

FireTVService

extends DeviceService

FireTVService provides capabilities for Amazon Fire TV and Fire TV Stick devices. FireTVService acts a layer on top of Amazon's Fling SDK, and requires the Fling SDK framework to function. FireTVService provides the following functionality:

- Media playback
- Media control

Using Connect SDK for discovery/control of Fire TV devices will result in your app complying with the Amazon Fling SDK terms of service.

Properties

id<BlockRunner> delegateBlockRunner The BlockRunner instance specifying where to run delegate callbacks. The default value is the main dispatch queue runner. Cannot be nil, as it will reset to the default value.

FireTVMediaPlayer * fireTVMediaPlayer Object that controls MediaPlayer functionality.

FireTVMediaControl * fireTVMediaControl Object that controls MediaControl functionality.

id<RemoteMediaPlayer> remoteMediaPlayer A RemoteMediaPlayer that's controlled by this service instance. It's returned from the ServiceDescription object, and thus can be nil if the serviceDescription property is nil.

AppStateChangeNotifier * appStateChangeNotifier An AppStateChangeNotifier that allows to track app state changes.

Methods

- (instancetype) **initWithAppStateChangeNotifier:(nullable *AppStateChangeNotifier* *)stateNotifier** Initializes the instance with the given *AppStateChangeNotifier*. Using *nil* parameter will create real object.

Parameters:

- stateNotifier

Inherited Methods

+ (NSDictionary *) **discoveryParameters** A dictionary of keys/values that will be used by the *DiscoveryProvider* used to discover this *DeviceService*. Some keys that are used are: service name, SSDP filter, etc.

+ (*DeviceService* *) **deviceServiceWithClass:(Class)_class serviceConfig:(ServiceConfig *)serviceConfig**

Parameters:

- _class
- serviceConfig: serviceConfig

+ (BOOL) **shouldDisconnectOnBackground** Static property that determines whether a *DeviceService* subclass should shut down communication channels when the app enters a background state. This may be helpful for apps that need to communicate with web apps from the background. This property may not be applicable to all *DeviceService* subclasses.

Sets the *shouldDisconnectOnBackground* static property. This property should be set before starting *DiscoveryManager* for the first time.

+ (void) **setShouldDisconnectOnBackground:(BOOL)shouldDisconnectOnBackground** **Parameters:**

- shouldDisconnectOnBackground

- (instancetype) **initWithServiceConfig:(ServiceConfig *)serviceConfig** **Parameters:**

- serviceConfig

- (BOOL) **hasCapability:(NSString *)capability** **Parameters:**

- capability

- (BOOL) **hasCapabilities:(NSArray *)capabilities** **Parameters:**

- capabilities

- (BOOL) **hasAnyCapability:(NSArray *)capabilities** **Parameters:**

- capabilities

- (void) **connect** Will attempt to connect to the *DeviceService*. The failure/success will be reported back to the *DeviceServiceDelegate*. If the connection attempt reveals that pairing is required, the *DeviceServiceDelegate* will also be notified in that event.

- (void) **disconnect** Will attempt to disconnect from the *DeviceService*. The failure/success will be reported back to the *DeviceServiceDelegate*.

- (void) **pairWithData:(id)pairingData** Will attempt to pair with the *DeviceService* with the provided *pairingData*. The failure/success will be reported back to the *DeviceServiceDelegate*.

Parameters:

- pairingData – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

- **(void) closeLaunchSession:(*LaunchSession* *)launchSession success:(*SuccessBlock*)success failure:(*FailureBlock*)failure**

Every LaunchSession object has an associated DeviceService. Internally, LaunchSession's close method proxies to its DeviceService's closeLaunchSession method. If, for some reason, your LaunchSession loses its DeviceService reference, you can call this closeLaunchSession method directly.

Parameters:

- **launchSession** – LaunchSession to be closed
- **success:** success – (optional) SuccessBlock to be called on success
- **failure:** failure – (optional) FailureBlock to be called on failure

NetcastTVService

extends DeviceService

NetcastTVService provides capabilities for LG Smart TVs running Netcast versions 3.x and 4.x (model years 2012-2014). The media playback functionality of NetcastTVService may be proxied through to DLNATVService to avoid requiring pairing. Commands & subscriptions on Netcast occur over HTTP.

The following capabilities are provided by the Netcast OS:

- Media playback
- Media control
- App launching*
- Volume control*
- Text input control*
- Key control (fiveway)*
- Mouse control*
- Power control*
- TV control (change channels, get channel info)*
- External input control*
- = requires pairing

To learn more about Netcast's second screen protocol, visit the [UDAP protocol specification](#).

Inherited Methods

+ (*NSDictionary* *) **discoveryParameters** A dictionary of keys/values that will be used by the DiscoveryProvider used to discover this DeviceService. Some keys that are used are: service name, SSDP filter, etc.

+ (*DeviceService* *) **deviceServiceWithClass:(Class)_class serviceConfig:(ServiceConfig *)serviceConfig**
Parameters:

- **_class**
- **serviceConfig:** serviceConfig

+ (**BOOL**) **shouldDisconnectOnBackground** Static property that determines whether a DeviceService subclass should shut down communication channels when the app enters a background state. This may be helpful for apps that need to communicate with web apps from the background. This property may not be applicable to all DeviceService subclasses.

Sets the `shouldDisconnectOnBackground` static property. This property should be set before starting Discovery-Manager for the first time.

+ (void) **setShouldDisconnectOnBackground:(BOOL)shouldDisconnectOnBackground** Parameters:

- `shouldDisconnectOnBackground`

- (instancetype) **initWithServiceConfig:(ServiceConfig *)serviceConfig** Parameters:

- `serviceConfig`

- (BOOL) **hasCapability:(NSString *)capability** Parameters:

- `capability`

- (BOOL) **hasCapabilities:(NSArray *)capabilities** Parameters:

- `capabilities`

- (BOOL) **hasAnyCapability:(NSArray *)capabilities** Parameters:

- `capabilities`

- (void) **connect** Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate. If the connection attempt reveals that pairing is required, the DeviceServiceDelegate will also be notified in that event.

- (void) **disconnect** Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate.

- (void) **pairWithData:(id)pairingData** Will attempt to pair with the DeviceService with the provided pairingData. The failure/success will be reported back to the DeviceServiceDelegate.

Parameters:

- `pairingData` – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

- (void) **closeLaunchSession:(LaunchSession *)launchSession success:(SuccessBlock)success failure:(FailureBlock)failure**

Every LaunchSession object has an associated DeviceService. Internally, LaunchSession's close method proxies to its DeviceService's closeLaunchSession method. If, for some reason, your LaunchSession loses its DeviceService reference, you can call this closeLaunchSession method directly.

Parameters:

- `launchSession` – LaunchSession to be closed
- **success:** success – (optional) SuccessBlock to be called on success
- **failure:** failure – (optional) FailureBlock to be called on failure

RokuService

extends DeviceService

RokuService provides many capabilities for Roku devices. Communication with Roku devices occurs over HTTP.

- List, launch, & close apps
- Media playback
- Media control
- Text input control
- Key control (fiveway)

These APIs should work on all Roku devices – they have been tested on Roku 2, Roku 3, and Roku Streaming Stick all running Roku 5.3 or later.

To learn more about the Roku External Control APIs, visit the [Roku External Control Guide](#).

Methods

+ (void) **registerApp:(NSString *)appId** **Parameters:**

- appId

Inherited Methods

+ (NSDictionary *) **discoveryParameters** A dictionary of keys/values that will be used by the DiscoveryProvider used to discover this DeviceService. Some keys that are used are: service name, SSDP filter, etc.

+ (*DeviceService* *) **deviceServiceWithClass:(Class)_class serviceConfig:(ServiceConfig *)serviceConfig** **Parameters:**

- _class
- **serviceConfig:** serviceConfig

+ (BOOL) **shouldDisconnectOnBackground** Static property that determines whether a DeviceService subclass should shut down communication channels when the app enters a background state. This may be helpful for apps that need to communicate with web apps from the background. This property may not be applicable to all DeviceService subclasses.

Sets the shouldDisconnectOnBackground static property. This property should be set before starting Discovery-Manager for the first time.

+ (void) **setShouldDisconnectOnBackground:(BOOL)shouldDisconnectOnBackground** **Parameters:**

- shouldDisconnectOnBackground

- (instancetype) **initWithServiceConfig:(ServiceConfig *)serviceConfig** **Parameters:**

- serviceConfig

- (BOOL) **hasCapability:(NSString *)capability** **Parameters:**

- capability

- (BOOL) **hasCapabilities:(NSArray *)capabilities** **Parameters:**

- capabilities

- (BOOL) **hasAnyCapability:(NSArray *)capabilities** **Parameters:**

- capabilities

- (void) **connect** Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate. If the connection attempt reveals that pairing is required, the DeviceServiceDelegate will also be notified in that event.

- (void) **disconnect** Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate.

- (void) **pairWithData:(id)pairingData** Will attempt to pair with the DeviceService with the provided pairingData. The failure/success will be reported back to the DeviceServiceDelegate.

Parameters:

- pairingData – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

- **(void) closeLaunchSession:(LaunchSession *)launchSession success:(SuccessBlock)success failure:(FailureBlock)failure**

Every LaunchSession object has an associated DeviceService. Internally, LaunchSession's close method proxies to its DeviceService's closeLaunchSession method. If, for some reason, your LaunchSession loses its DeviceService reference, you can call this closeLaunchSession method directly.

Parameters:

- launchSession – LaunchSession to be closed
- **success:** success – (optional) SuccessBlock to be called on success
- **failure:** failure – (optional) FailureBlock to be called on failure

WebOSTVService

extends DeviceService

WebOSTVService provides capabilities for LG Smart TVs running webOS (model year 2014). The second screen gateway running on the webOS provides different capabilities based on whether pairing is enabled or not.

- Web app launching & two-way communication
- App launching
- Media playback
- Media control
- Volume control
- Text input control*
- Key control (fiveway)*
- Mouse control*
- Power control*
- TV control (change channels, get channel info)*
- External input control*
- Toast control*

* = requires pairing

Commands & subscriptions on webOS occur over a WebSocket connection.

webOS Version History

The following version numbers represent the version of webOS released for LG Smart TVs. The version numbers are associated with any changes to the platform's second screen APIs in that particular version.

4.0.0

- Initial release

4.0.1

- No changes

4.0.2

- Added app-to-app support
- Added the ability to request pin or prompt pairing

4.0.3

- Fixed a subscription bug in app-to-app

Inherited Methods

+ (**NSDictionary** *) **discoveryParameters** A dictionary of keys/values that will be used by the DiscoveryProvider used to discover this DeviceService. Some keys that are used are: service name, SSDP filter, etc.

+ (**DeviceService** *) **deviceServiceWithClass:(Class)_class serviceConfig:(ServiceConfig *)serviceConfig**
Parameters:

- **_class**
- **serviceConfig:** serviceConfig

+ (**BOOL**) **shouldDisconnectOnBackground** Static property that determines whether a DeviceService subclass should shut down communication channels when the app enters a background state. This may be helpful for apps that need to communicate with web apps from the background. This property may not be applicable to all DeviceService subclasses.

Sets the shouldDisconnectOnBackground static property. This property should be set before starting Discovery-Manager for the first time.

+ (**void**) **setShouldDisconnectOnBackground:(BOOL)shouldDisconnectOnBackground** **Parameters:**

- shouldDisconnectOnBackground

- (**instancetype**) **initWithServiceConfig:(ServiceConfig *)serviceConfig** **Parameters:**

- serviceConfig

- (**BOOL**) **hasCapability:(NSString *)capability** **Parameters:**

- capability

- (**BOOL**) **hasCapabilities:(NSArray *)capabilities** **Parameters:**

- capabilities

- (**BOOL**) **hasAnyCapability:(NSArray *)capabilities** **Parameters:**

- capabilities

- (**void**) **connect** Will attempt to connect to the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate. If the connection attempt reveals that pairing is required, the DeviceServiceDelegate will also be notified in that event.

- (**void**) **disconnect** Will attempt to disconnect from the DeviceService. The failure/success will be reported back to the DeviceServiceDelegate.

- (**void**) **pairWithData:(id)pairingData** Will attempt to pair with the DeviceService with the provided pairingData. The failure/success will be reported back to the DeviceServiceDelegate.

Parameters:

- pairingData – Data to be used for pairing. The type of this parameter will vary depending on what type of pairing is required, but is likely to be a string (pin code, pairing key, etc).

- **(void) closeLaunchSession:(*LaunchSession* *)launchSession success:(*SuccessBlock*)success failure:(*FailureBlock*)failure**

Every LaunchSession object has an associated DeviceService. Internally, LaunchSession's close method proxies to its DeviceService's closeLaunchSession method. If, for some reason, your LaunchSession loses its DeviceService reference, you can call this closeLaunchSession method directly.

Parameters:

- launchSession – LaunchSession to be closed
- **success:** success – (optional) SuccessBlock to be called on success
- **failure:** failure – (optional) FailureBlock to be called on failure

5.16.4 Capabilities

CapabilityPriorityLevel

CapabilityPriorityLevel values are used by ConnectableDevice to find the most suitable DeviceService capability to be presented to the user. Values of VeryLow and VeryHigh are not in use internally the SDK. Connect SDK uses Low, Normal, and High internally.

Default behavior: If you are unsatisfied with the default priority levels & behavior of Connect SDK, it is possible to subclass a particular DeviceService and provide your own value for each capability. That DeviceService subclass would need to be registered with DiscoveryManager.

Properties

CapabilityPriorityLevelVeryLow

CapabilityPriorityLevelLow

CapabilityPriorityLevelNormal

CapabilityPriorityLevelHigh

CapabilityPriorityLevelVeryHigh

ExternalInputControl

The ExternalInputControl capability serves to define the methods required for normalizing all functions regarding external input switching and general info.

Methods

- (id<*ExternalInputControl*>) **externalInputControl**

- (*CapabilityPriorityLevel*) **externalInputControlPriority**

- **(void) launchInputPickerWithSuccess:(*AppLaunchSuccessBlock*)success failure:(*FailureBlock*)failure**

Launches the visual input picker on the device. This may be helpful for situations where the device does not support directly listing/modifying the external inputs.

Related capabilities:

- `ExternalInputControl.Picker.Launch`

Parameters:

- success – Optional AppLaunchSuccessBlock to be called on success
 - **failure:** failure – Optional FailureBlock to be called on failure
- (void) **closeInputPicker:***(LaunchSession *)launchSession success:(SuccessBlock)success failure:(FailureBlock)failure*
Closes the input picker on the device, if it is currently open.

Related capabilities:

- ExternalInputControl.Picker.Close

Parameters:

- launchSession – LaunchSession from the ExternalInputListSuccessBlock
 - **success:** success – Optional SuccessBlock to be called on success
 - **failure:** failure – Optional FailureBlock to be called on failure
- (void) **getExternalInputListWithSuccess:***(ExternalInputListSuccessBlock)success failure:(FailureBlock)failure*
Get a list of input devices (HDMI, AV, etc) connected to the device

Related capabilities:

- ExternalInputControl.List

Parameters:

- success – Optional ExternalInputListSuccessBlock to be called on success
 - **failure:** failure – Optional FailureBlock to be called on failure
- (void) **setExternalInput:***(ExternalInputInfo *)externalInputInfo success:(SuccessBlock)success failure:(FailureBlock)failure*
Switch to the specified external input

Related capabilities:

- ExternalInputControl.Set

Parameters:

- externalInputInfo – Object containing the proper info to set current input. For best cross-platform support, it is suggested to get ExternalInputInfo references from getExternalInputList, if possible.
- **success:** success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

Typedefs

ExternalInputListSuccessBlock

void(^)(NSArray *externalInputList)

Success block that is called upon successfully getting the external input list.

- externalInputList

Array containing an ExternalInputInfo object for each available external input on the device

KeyControl

The KeyControl capability serves to define the methods required for normalizing common key commands (up, down, left right, ok, back, home, key code).

Methods

- (id<*KeyControl*>) **keyControl**

- (*CapabilityPriorityLevel*) **keyControlPriority**

- (void) **upWithSuccess:(SuccessBlock)*success* failure:(FailureBlock)*failure*** Sends the up button key code to the TV.

Related capabilities:

- `KeyControl.Up`

Parameters:

name parameters

class method-detail-label

- **success** – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **downWithSuccess:(SuccessBlock)*success* failure:(FailureBlock)*failure*** Sends the down button key code to the TV.

Related capabilities:

- `KeyControl.Down`

Parameters:

- **success** – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **leftWithSuccess:(SuccessBlock)*success* failure:(FailureBlock)*failure*** Sends the left button key code to the TV.

Related capabilities:

- `KeyControl.Left`

Parameters:

- **success** – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **rightWithSuccess:(SuccessBlock)*success* failure:(FailureBlock)*failure*** Sends the right button key code to the TV.

Related capabilities:

- `KeyControl.Right`

Parameters:

- **success** – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **okWithSuccess:(SuccessBlock)*success* failure:(FailureBlock)*failure*** Sends the OK button key code to the TV.

Related capabilities:

- `KeyControl.OK`

Parameters:

- success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **backWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Sends the back button key code to the TV.

Related capabilities:

- `KeyControl.Back`

Parameters:

- success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **homeWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Sends the home button key code to the TV.

Related capabilities:

- `KeyControl.Home`

Parameters:

- success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **sendKeyCode:(NSInteger)keyCode success:(SuccessBlock)success failure:(FailureBlock)failure**
Sends a key code value to the TV.

Related capabilities:

- `KeyControl.Send.KeyCode`

Parameters:

- keyCode
- **success:** success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

Launcher

The Launcher capability protocol serves to define the methods required for normalizing the launching of apps. It allows for in-built support for certain common launch types (deep-linking to YouTube, Netflix, Hulu, browser, etc) as well as by (platform-specific) app id.

Methods

- (id<Launcher>) **launcher**

- (CapabilityPriorityLevel) **launcherPriority**

- (void) **launchApp:(NSString *)appId success:(AppLaunchSuccessBlock)success failure:(FailureBlock)failure**
Launch an application on the device.

Related capabilities:

- `Launcher.App`

Parameters:

- `appId` – ID of the application
- **success:** success – Optional `AppLaunchSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **launchAppWithInfo:(AppInfo *)*appInfo* success:(AppLaunchSuccessBlock)*success* failure:(FailureBlock)*failure***
Launch an application on the device.

Related capabilities:

- `Launcher.App`
- `Launcher.App.Params` – if launching with params

Parameters:

- `appInfo` – `AppInfo` object for the application
- **success:** success – Optional `AppLaunchSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **launchAppWithInfo:(AppInfo *)*appInfo* params:(NSDictionary *)*params* success:(AppLaunchSuccessBlock)*success* failure:(FailureBlock)*failure***
Launch an application on the device.

Related capabilities:

- `Launcher.App`
- `Launcher.App.Params` – if launching with params

Parameters:

- `appInfo` – `AppInfo` object for the application
- **params:** params
- **success:** success – Optional `AppLaunchSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **closeApp:(LaunchSession *)*launchSession* success:(SuccessBlock)*success* failure:(FailureBlock)*failure***
Close an application on the device.

Related capabilities:

- `Launcher.App.Close`

Parameters:

- `launchSession` – `LaunchSession` of the target app
- **success:** success – Optional `SuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **getAppListWithSuccess:(AppListSuccessBlock)*success* failure:(FailureBlock)*failure*** Gets a list of all apps installed on the device.

Related capabilities:

- `Launcher.App.List`

Parameters:

- **success** – Optional `AppListSuccessBlock` to be called on success

- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **getRunningAppWithSuccess:(AppInfoSuccessBlock)success failure:(FailureBlock)failure** Gets an AppInfo object for the current running app on the device.

Related capabilities:

- Launcher.RunningApp

Parameters:

- **success** – Optional AppInfoSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (*ServiceSubscription* *) **subscribeRunningAppWithSuccess:(AppInfoSuccessBlock)success failure:(FailureBlock)failure**
Subscribes to changes of the current running app. Every time the running app changes, the success block will be called with an AppInfo object for the current running app.

Related capabilities:

- Launcher.RunningApp.Subscribe

Parameters:

- **success** – Optional AppInfoSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **getAppState:(LaunchSession *)launchSession success:(AppStateSuccessBlock)success failure:(FailureBlock)failure**
Gets the target app's running status and on-screen visibility.

Related capabilities:

- Launcher.AppState

Parameters:

- **launchSession** – LaunchSession of the target app
- **success:** success – Optional AppStateSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (*ServiceSubscription* *) **subscribeAppState:(LaunchSession *)launchSession success:(AppStateSuccessBlock)success failure:(FailureBlock)failure**

Subscribes to changes of the state of the target app. Every time the app's state changes, the success block will be called with info on the app's running status and on-screen visibility.

Related capabilities:

- Launcher.AppState.Subscribe

Parameters:

- **launchSession** – LaunchSession of the target app
- **success:** success – Optional AppStateSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **launchAppStore:(NSString *)appId success:(AppLaunchSuccessBlock)success failure:(FailureBlock)failure**
Launch the device's app store app, optionally deep-linked to a specific app's page.

Related capabilities:

- Launcher.AppStore

- `Launcher.AppStore.Params`

Parameters:

- `appId` – (optional) ID of the application to show in the app store
- **success:** success – Optional `AppLaunchSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **launchBrowser:(NSURL *)target success:(AppLaunchSuccessBlock)success failure:(FailureBlock)failure**
Launch the web browser. Will launch deep-linked to provided URL, if supported on the target platform.

Related capabilities:

- `Launcher.Browser`
- `Launcher.Browser.Params` – if launching with url

Parameters:

- `target` – URL to open
- **success:** success – Optional `AppLaunchSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **launchYouTube:(NSString *)contentId success:(AppLaunchSuccessBlock)success failure:(FailureBlock)failure**
Launch YouTube app. Will launch deep-linked to provided contentId, if supported on the target platform.

Related capabilities:

- `Launcher.YouTube`
- `Launcher.YouTube.Params` – if launching with contentId

Parameters:

- `contentId` – Video id to open
- **success:** success – Optional `AppLaunchSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **launchYouTube:(NSString *)contentId startTime:(float)startTime success:(AppLaunchSuccessBlock)success failure:(FailureBlock)failure**
Launch YouTube app. Will launch deep-linked to provided contentId, if supported on the target platform.

Related capabilities:

- `Launcher.YouTube`
- `Launcher.YouTube.Params` – if launching with contentId

Parameters:

- `contentId` – Video id to open
- **startTime:** startTime
- **success:** success – Optional `AppLaunchSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

Typedefs

AppInfoSuccessBlock

`void(^)(AppInfo *appInfo)`

Success block that is called upon requesting info about the current running app.

- `appInfo`
Object containing info about the running app

AppLaunchSuccessBlock

`void(^)(LaunchSession *launchSession)`

Success block that is called upon successfully launching an app.

AppListSuccessBlock

`void(^)(NSArray *appList)`

Success block that is called upon successfully getting the app list.

- `appList`
Array containing an AppInfo object for each available app on the device

AppStateSuccessBlock

`void(^)(BOOL running, BOOL visible)`

Success block that is called upon successfully getting an app's state.

- `running`
Whether the app is currently running
- `visible`
Whether the app is currently visible on the screen

MediaControl

The MediaControl capability protocol serves to define the methods required for normalizing the control of media playback (play, pause, fast forward, etc) as well as obtaining media information (playhead position, duration, etc).

Methods

- (id<MediaControl>) **mediaControl**

- (CapabilityPriorityLevel) **mediaControlPriority**

- (void) **playWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Send play command.

Related capabilities:

- `MediaControl.Play`

Parameters:

- **success** – Optional `SuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **pauseWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Send pause command.

Related capabilities:

- `MediaControl.Pause`

Parameters:

- **success** – Optional `SuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **stopWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Send play command.

Related capabilities:

- `MediaControl.Stop`

Parameters:

- **success** – Optional `SuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **rewindWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Send rewind command.

Related capabilities:

- `MediaControl.Rewind`

Parameters:

- **success** – Optional `SuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **fastForwardWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Send play command.

Related capabilities:

- `MediaControl.FastForward`

Parameters:

- **success** – Optional `SuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **seek:(NSTimeInterval)position success:(SuccessBlock)success failure:(FailureBlock)failure** Seeks to a new position within the current media item

Related capabilities:

- `MediaControl.Seek`

Parameters:

- **position**
- **success:** success – Optional `SuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **getDurationWithSuccess:***(MediaDurationSuccessBlock)***success failure:(FailureBlock)failure**
Parameters:

- **success** – Optional MediaDurationSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **getPositionWithSuccess:***(MediaPositionSuccessBlock)***success failure:(FailureBlock)failure**
Parameters:

- **success** – Optional MediaPositionSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **getMediaMetaDataWithSuccess:***(SuccessBlock)***success failure:(FailureBlock)failure** **Parameters:**

- **success** – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **getPlayStateWithSuccess:***(MediaPlayStateSuccessBlock)***success failure:(FailureBlock)failure**
Parameters:

- **success** – Optional MediaPlayStateSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (*ServiceSubscription* *) **subscribePlayStateWithSuccess:***(MediaPlayStateSuccessBlock)***success failure:(FailureBlock)failure**
Parameters:

- **success** – Optional MediaPlayStateSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (*ServiceSubscription* *) **subscribeMediaInfoWithSuccess:***(SuccessBlock)***success failure:(FailureBlock)failure**
Parameters:

- **success** – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

Typedefs

MediaPlayStateSuccessBlock

void(^)(*MediaControlPlayState* playState)

Success block that is called upon any change in a media file's play state.

- **playState**
Play state of the current media file

MediaPositionSuccessBlock

void(^)(NSTimeInterval position)

Success block that is called upon successfully getting the media file's current playhead position.

- **position**
Current playhead position of the current media file, in seconds

MediaDurationSuccessBlock

void(^)(NSTimeInterval duration)

Success block that is called upon successfully getting the media file's duration.

- duration
Duration of the current media file, in seconds

MediaPlayer

The MediaPlayer capability protocol serves to define the methods required for displaying media on the device.

Methods

- (id<MediaPlayer>) **mediaPlayer**

- (*CapabilityPriorityLevel*) **mediaPlayerPriority**

- (void) **displayImageWithMediaInfo:(MediaInfo *)mediaInfo success:(MediaPlayerSuccessBlock)success failure:(FailureBlock)failure**

Parameters:

- mediaInfo
- **success:** success – Optional MediaPlayerSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **playMediaWithMediaInfo:(MediaInfo *)mediaInfo shouldLoop:(BOOL)shouldLoop success:(MediaPlayerSuccessBlock)success failure:(FailureBlock)failure**

Parameters:

- mediaInfo
- **shouldLoop:** shouldLoop
- **success:** success – Optional MediaPlayerSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **closeMedia:(LaunchSession *)launchSession success:(SuccessBlock)success failure:(FailureBlock)failure**

Close a running media session. Because media is handled differently on different platforms, it is required to keep track of LaunchSession and MediaControl objects to control that media session in the future. LaunchSession will be required to close the media and mediaControl will be required to control the media.

Related capabilities:

- MediaPlayer.Close

Parameters:

- launchSession – LaunchSession object for use in closing media instance
- **success:** success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **displayImage:(NSURL *)imageURL iconURL:(NSURL *)iconURL title:(NSString *)title description:(NSString *)description**

Display an image on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- `MediaPlayer.Display.Image`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- `imageURL` – URL of image to open
- **iconURL:** `iconURL` – URL of an icon to show next to the title
- **title:** `title` – Title text to display
- **description:** `description` – Description text to display
- **contentType:** `contentType` – MIME type of the image, for example “image/jpeg”
- **success:** `success` – Optional `MediaPlayerDisplaySuccessBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

- (void) **displayImage:(MediaInfo *)mediaInfo success:(MediaPlayerDisplaySuccessBlock)success failure:(FailureBlock)failure**
Display an image on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- `MediaPlayer.Display.Image`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- `mediaInfo` – Object of `MediaInfo` class which includes all the information about an image to display.
- **success:** `success` – Optional `MediaPlayerDisplaySuccessBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

- (void) **playMedia:(NSURL *)mediaURL iconURL:(NSURL *)iconURL title:(NSString *)title description:(NSString *)description**
Play an audio or video file on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- `MediaPlayer.Play.Video`
- `MediaPlayer.Play.Audio`
- `MediaPlayer.MediaData.Title`
- `MediaPlayer.MediaData.Description`
- `MediaPlayer.MediaData.Thumbnail`
- `MediaPlayer.MediaData.MimeType`

Parameters:

- **mediaURL** – URL of media file to open
- **iconURL**: iconURL – URL of an icon to show next to the title
- **title**: title – Title text to display
- **description**: description – Description text to display
- **mimeType**: mimeType – MIME type of the video, for example “video/mpeg4”, “audio/mp3”, etc
- **shouldLoop**: shouldLoop – Whether to automatically loop playback
- **success**: success – Optional MediaPlayerDisplaySuccessBlock to be called on success
- **failure**: failure – Optional FailureBlock to be called on failure

- (void) **playMedia:(MediaInfo *)mediaInfo shouldLoop:(BOOL)shouldLoop success:(MediaPlayerDisplaySuccessBlock)success failure:(FailureBlock)failure**
 Play an audio or video file on the device. Not all devices support all of the parameters – supply as many as you have available.

Related capabilities:

- MediaPlayer.Play.Video
- MediaPlayer.Play.Audio
- MediaPlayer.MediaData.Title
- MediaPlayer.MediaData.Description
- MediaPlayer.MediaData.Thumbnail
- MediaPlayer.MediaData.MimeType

Parameters:

- **mediaInfo** – Object of MediaInfo class which includes all the information about an image to display.
- **shouldLoop**: shouldLoop – Whether to automatically loop playback
- **success**: success – Optional MediaPlayerDisplaySuccessBlock to be called on success
- **failure**: failure – Optional FailureBlock to be called on failure

Typedefs

MediaPlayerDisplaySuccessBlock

```
void(^)(LaunchSession *launchSession, id<MediaControl> mediaControl)
```

Success block that is called upon successfully playing/displaying a media file.

- **launchSession**
LaunchSession to allow closing this media player
- **mediaControl**
MediaControl object used to control playback

MediaPlayerSuccessBlock

```
void(^)(MediaLaunchObject *mediaLaunchObject)
```

MouseControl

The MouseControl capability serves to define the methods required for normalizing a mouse/trackpad (move/scroll with relative coordinates and click).

Methods

- (id<MouseControl>) **mouseControl**

- (*CapabilityPriorityLevel*) **mouseControlPriority**

- **(void) connectMouseWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Establish a connection with the DeviceService's mouse communication medium (WebSocket, HTTP, etc). While this step may not be necessary with certain platforms, it is suggested to call it anyways, for purposes of seamless normalization. Calling connect on a non-connectable protocol will just trigger the success callback immediately.

Related capabilities:

- `MouseControl.Connect`

Parameters:

- **success** – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- **(void) disconnectMouse** Disconnects from the mouse communication medium.

Related capabilities:

- `MouseControl.Disconnect`

- **(void) clickWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Perform a click action at the current mouse position.

Related capabilities:

- `MouseControl.Click`

Parameters:

- **success** – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- **(void) move:(CGVector)distance success:(SuccessBlock)success failure:(FailureBlock)failure** Move the mouse by the given distance values.

Related capabilities:

- `MouseControl.Move`

Parameters:

- **distance** – Distance to move the mouse relative to its current position
- **success:** success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- **(void) scroll:(CGVector)distance success:(SuccessBlock)success failure:(FailureBlock)failure** Scroll by the given distance values.

Related capabilities:

- `MouseControl.Scroll`

Parameters:

- **distance** – Distance to scroll relative to current position
- **success**: success – Optional SuccessBlock to be called on success
- **failure**: failure – Optional FailureBlock to be called on failure

PlayListControl**Methods**

- (id<*PlayListControl*>) **playListControl**

- (*CapabilityPriorityLevel*) **playListControlPriority**

- (void) **playNextWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Plays the next track in the playlist

Parameters:

- **success** – Optional SuccessBlock to be called on success
- **failure**: failure – Optional FailureBlock to be called on failure

- (void) **playPreviousWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Plays the previous track in the playlist

Parameters:

- **success** – Optional SuccessBlock to be called on success
- **failure**: failure – Optional FailureBlock to be called on failure

- (void) **jumpToTrackWithIndex:(NSInteger)index success:(SuccessBlock)success failure:(FailureBlock)failure** Jumps to track in the playlist

Parameters:

- **index** – NSInteger a zero based index parameter.
- **success**: success – Optional SuccessBlock to be called on success
- **failure**: failure – Optional FailureBlock to be called on failure

PowerControl

The PowerControl capability protocol serves to define the methods required for normalizing power off functionality.

Methods

- (id<*PowerControl*>) **powerControl**

- (*CapabilityPriorityLevel*) **powerControlPriority**

- (void) **powerOffWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure**

Sends a power off signal to the TV. A success message will, internally, trigger a disconnection with the device.

Related capabilities:

- `PowerControl.Off`

Parameters:

- `success` – Optional `SuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **powerOnWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** **Parameters:**

- `success` – Optional `SuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

TVControl

The `TVControl` capability protocol serves to define the methods required for normalizing common TV-specific commands (channel up/down, channel list, channel info, etc).

Methods

- (id<*TVControl*>) **tvControl**

- (*CapabilityPriorityLevel*) **tvControlPriority**

- (void) **channelUpWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Sends a channel up command to the TV.

Related capabilities:

- `TVControl.Channel.Up`

Parameters:

- `success` – Optional `SuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **channelDownWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Sends a channel down command to the TV.

Related capabilities:

- `TVControl.Channel.Down`

Parameters:

- `success` – Optional `SuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **setChannel:(ChannelInfo *)channelInfo success:(SuccessBlock)success failure:(FailureBlock)failure** Sets the current channel to the channel provided by the `ChannelInfo` object provided.

Related capabilities:

- `TVControl.Channel.Set`

Parameters:

- `channelInfo` – `ChannelInfo` object containing information about the desired channel
- **success:** success – Optional `SuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **getCurrentChannelWithSuccess:(*CurrentChannelSuccessBlock*)success failure:(FailureBlock)failure**
Gets the current channel info from the TV.

Related capabilities:

- `TVControl.Channel.Get`

Parameters:

- **success** – Optional `CurrentChannelSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (*ServiceSubscription* *) **subscribeCurrentChannelWithSuccess:(*CurrentChannelSuccessBlock*)success failure:(FailureBlock)failure**
Subscribes to any changes in the current channel. Each time the channel is changed, the new channel's info will be provided to the success callback.

Related capabilities:

- `TVControl.Channel.Subscribe`

Parameters:

- **success** – Optional `CurrentChannelSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **getChannelListWithSuccess:(*ChannelListSuccessBlock*)success failure:(FailureBlock)failure** Get a list of available channels from the TV.

Related capabilities:

- `TVControl.Channel.List`

Parameters:

- **success** – Optional `ChannelListSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **getProgramInfoWithSuccess:(*ProgramInfoSuccessBlock*)success failure:(FailureBlock)failure** Gets the current program info from the TV.

Related capabilities:

- `TVControl.Program.Get`

Parameters:

- **success** – Optional `ProgramInfoSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (*ServiceSubscription* *) **subscribeProgramInfoWithSuccess:(*ProgramInfoSuccessBlock*)success failure:(FailureBlock)failure**
Subscribes to any changes in the current program. Each time the channel is changed or a new program starts, the new program's info will be provided to the success callback.

Related capabilities:

- `TVControl.Program.Subscribe`

Parameters:

- **success** – Optional `ProgramInfoSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **getProgramListWithSuccess:(*ProgramListSuccessBlock*)success failure:(FailureBlock)failure** Gets a list of all programs scheduled to play on the current channel.

Related capabilities:

- `TVControl.Program.List`

Parameters:

- **success** – Optional `ProgramListSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (*ServiceSubscription* *) **subscribeProgramListWithSuccess:(*ProgramListSuccessBlock*)success failure:(FailureBlock)failure**
Subscribes to any changes in the current program. Each time the channel is changed or a new program starts, the new program's info will be provided to the success callback.

Related capabilities:

- `TVControl.Program.List.Subscribe`

Parameters:

- **success** – Optional `ProgramListSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **get3DEnabledWithSuccess:(*TV3DEnabledSuccessBlock*)success failure:(FailureBlock)failure** Gets the current 3D status of the TV.

Related capabilities:

- `TVControl.3D.Get`

Parameters:

- **success** – Optional `TV3DEnabledSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (void) **set3DEnabled:(BOOL)enabled success:(SuccessBlock)success failure:(FailureBlock)failure** Sets the current 3D status of the TV.

Related capabilities:

- `TVControl.3D.Set`

Parameters:

- **enabled** – Whether the TV's 3D mode should be on or off
- **success:** success – Optional `SuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

- (*ServiceSubscription* *) **subscribe3DEnabledWithSuccess:(*TV3DEnabledSuccessBlock*)success failure:(FailureBlock)failure**
Subscribes to changes in the TV's 3D status.

Related capabilities:

- `TVControl.3D.Subscribe`

Parameters:

- **success** – Optional `TV3DEnabledSuccessBlock` to be called on success
- **failure:** failure – Optional `FailureBlock` to be called on failure

Typedefs

CurrentChannelSuccessBlock

```
void(^)(ChannelInfo *channelInfo)
```

Success block that is called upon successfully getting the current channel's information.

- channelInfo
Object containing information about the current channel

ChannelListSuccessBlock

```
void(^)(NSArray *channelList)
```

Success block that is called upon successfully getting the channel list.

- channelList
Array containing a ChannelInfo object for each available channel on the TV

ProgramInfoSuccessBlock

```
void(^)(ProgramInfo *programInfo)
```

Success block that is called upon successfully getting the current program's information.

- programInfo
Object containing information about the current program

ProgramListSuccessBlock

```
void(^)(NSArray *programList)
```

Success block that is called upon successfully getting the program list for the current channel.

- programList
Array containing a ProgramInfo object for each available program on the TV's current channel

TV3DEnabledSuccessBlock

```
void(^)(BOOL tv3DEnabled)
```

Success block that is called upon successfully getting the TV's 3D mode

- tv3DEnabled
Whether 3D mode is currently enabled on the TV

TextInputControl

The TextInputControl capability serves to define the methods required for normalizing common text input commands (send text, enter, delete, keyboard status).

Methods

- (id<TextInputControl>) **textInputControl**

- (CapabilityPriorityLevel) **textInputControlPriority**

- (ServiceSubscription *) **subscribeTextInputStatusWithSuccess:(TextInputStatusInfoSuccessBlock)success failure:(FailureBlock)failure**
Subscribe to information about the current text field.

Related capabilities:

- `TextInputControl.Subscribe`

Parameters:

- **success** – Optional `TextInputStatusInfoSuccessBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

- (void) **sendText:(NSString *)input success:(SuccessBlock)success failure:(FailureBlock)failure** Send text to the current text field.

Related capabilities:

- `TextInputControl.Send.Text`

Parameters:

- **input**
- **success:** `success` – Optional `SuccessBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

- (void) **sendEnterWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Send enter key to the current text field.

Related capabilities:

- `TextInputControl.Send.Enter`

Parameters:

- **success** – Optional `SuccessBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

- (void) **sendDeleteWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Send delete event to the current text field.

Related capabilities:

- `TextInputControl.Send.Delete`

Parameters:

- **success** – Optional `SuccessBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

Typedefs

TextInputStatusInfoSuccessBlock

`void (^)(TextInputStatusInfo *textInputStatusInfo)`

Response block that is fired on any change of keyboard visibility.

- `textInputStatusInfo`
provides keyboard type & visibility information

ToastControl

The ToastControl capability protocol serves to define the methods required for displaying toast messages on the TV.

Toasts may optionally provide an 80x80 pixel icon in PNG or JPEG format, encoded as base64. The icon will be displayed alongside the toast message.

Methods

- (id<*ToastControl*>) **toastControl**

- (*CapabilityPriorityLevel*) **toastControlPriority**

- (void) **showToast:(NSString *)message success:(SuccessBlock)success failure:(FailureBlock)failure** Show a toast on the TV.

Parameters:

- `message` – Message to display
- **success:** success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **showToast:(NSString *)message iconData:(NSString *)iconData iconExtension:(NSString *)iconExtension success:(SuccessBlock)success failure:(FailureBlock)failure** Show a toast on the TV.

Parameters:

- `message` – Message to display
- **iconData:** iconData – Base-64 encoded JPEG or PNG data
- **iconExtension:** iconExtension – File extension of icon
- **success:** success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **showClickableToast:(NSString *)message appInfo:(AppInfo *)appInfo params:(NSDictionary *)params success:(SuccessBlock)success failure:(FailureBlock)failure** Show a toast on the TV and perform an action when the toast is clicked on the TV.

Related capabilities:

- `ToastControl.Show.Clickable.App`
- `ToastControl.Show.Clickable.App.Params`
- `ToastControl.Show.Clickable.URL`

Parameters:

- `message` – Message to display
- **appInfo:** appInfo – AppInfo for app to launch on click of toast
- **params:** params – Launch params for app
- **success:** success – Optional SuccessBlock to be called on success

- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **showClickableToast:(NSString *)message appInfo:(AppInfo *)appInfo params:(NSDictionary *)params iconData:(NSData *)iconData**
Show a toast on the TV and perform an action when the toast is clicked on the TV.

Related capabilities:

- ToastControl.Show.Clickable.App
- ToastControl.Show.Clickable.App.Params
- ToastControl.Show.Clickable.URL

Parameters:

- **message** – Message to display
- **appInfo:** appInfo – AppInfo for app to launch on click of toast
- **params:** params – Launch params for app
- **iconData:** iconData – Base-64 encoded JPEG or PNG data
- **iconExtension:** iconExtension – File extension of icon
- **success:** success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **showClickableToast:(NSString *)message URL:(NSURL *)URL success:(SuccessBlock)success failure:(FailureBlock)failure**
Show a toast on the TV and perform an action when the toast is clicked on the TV.

Related capabilities:

- ToastControl.Show.Clickable.App
- ToastControl.Show.Clickable.App.Params
- ToastControl.Show.Clickable.URL

Parameters:

- **message** – Message to display
- **URL:** URL – URL to launch in browser
- **success:** success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **showClickableToast:(NSString *)message URL:(NSURL *)URL iconData:(NSData *)iconData iconExtension:(NSString *)iconExtension**
Show a toast on the TV and perform an action when the toast is clicked on the TV.

Related capabilities:

- ToastControl.Show.Clickable.App
- ToastControl.Show.Clickable.App.Params
- ToastControl.Show.Clickable.URL

Parameters:

- **message** – Message to display
- **URL:** URL – URL to launch in browser
- **iconData:** iconData – Base-64 encoded JPEG or PNG data
- **iconExtension:** iconExtension – File extension of icon

- **success:** success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

VolumeControl

The VolumeControl capability protocol serves to define the methods required for normalizing common volume specific commands (volume up/down, mute, etc).

Methods

- (id<*VolumeControl*>) **volumeControl**

- (*CapabilityPriorityLevel*) **volumeControlPriority**

- (void) **volumeUpWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Sends the volume up command to the device.

Related capabilities:

- `VolumeControl.UpDown`

Parameters:

- **success** – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **volumeDownWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Sends the volume down command to the device.

Related capabilities:

- `VolumeControl.UpDown`

Parameters:

- **success** – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **getVolumeWithSuccess:(VolumeSuccessBlock)success failure:(FailureBlock)failure** Get the current volume of the device.

Related capabilities:

- `VolumeControl.Get`

Parameters:

- **success** – Optional VolumeSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **setVolume:(float)volume success:(SuccessBlock)success failure:(FailureBlock)failure** Set the volume of the device.

Related capabilities:

- `VolumeControl.Set`

Parameters:

- **volume** – Volume as a float between 0.0 and 1.0

- **success:** success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (*ServiceSubscription* *) **subscribeVolumeWithSuccess:(VolumeSuccessBlock)success failure:(FailureBlock)failure**
Subscribe to the volume on the TV.

Related capabilities:

- `VolumeControl.Subscribe`

Parameters:

- **success** – Optional VolumeSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (**void**) **getMuteWithSuccess:(MuteSuccessBlock)success failure:(FailureBlock)failure** Get the current mute state.

Related capabilities:

- `VolumeControl.Mute.Get`

Parameters:

- **success** – Optional MuteSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (**void**) **setMute:(BOOL)mute success:(SuccessBlock)success failure:(FailureBlock)failure** Set the current volume.

Related capabilities:

- `VolumeControl.Mute.Set`

Parameters:

- **mute**
- **success:** success – Optional SuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (*ServiceSubscription* *) **subscribeMuteWithSuccess:(MuteSuccessBlock)success failure:(FailureBlock)failure**
Subscribe to the mute state on the TV.

Related capabilities:

- `VolumeControl.Mute.Subscribe`

Parameters:

- **success** – Optional MuteSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

Typedefs

VolumeSuccessBlock

`void(^)(float volume)`

Success block that is called upon successfully getting the device's system volume.

- volume

Current system volume, value is a float between 0.0 and 1.0

MuteSuccessBlock

void(^)(BOOL mute)

Success block that is called upon successfully getting the device's system mute status.

- mute

Current system mute status

WebAppLauncher

The WebAppLauncher capability protocol provides capabilities for launching web apps and establishing two-way communication.

Methods

- (id<WebAppLauncher>) **webAppLauncher**

- (*CapabilityPriorityLevel*) **webAppLauncherPriority**

- (void) **launchWebApp:(NSString *)webAppId success:(WebAppLaunchSuccessBlock)success failure:(FailureBlock)failure**
Launch a web application on the TV.

Related capabilities:

- WebAppLauncher.Launch
- WebAppLauncher.Launch.Params – if launching with params

Parameters:

- webAppId – ID of web app assigned by platform vendor
- **success:** success – Optional WebAppLaunchSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **launchWebApp:(NSString *)webAppId params:(NSDictionary *)params success:(WebAppLaunchSuccessBlock)success failure:(FailureBlock)failure**
Launch a web application on the TV.

Related capabilities:

- WebAppLauncher.Launch
- WebAppLauncher.Launch.Params – if launching with params

Parameters:

- webAppId – ID of web app assigned by platform vendor
- **params:** params – Dictionary of key/value strings. Not available on all target platforms
- **success:** success – Optional WebAppLaunchSuccessBlock to be called on success
- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **launchWebApp:(NSString *)webAppId relaunchIfRunning:(BOOL)relaunchIfRunning success:(WebAppLaunchSuccessBlock)success**
Launch a web application on the TV.

This method requires pairing on webOS

Related capabilities:

- `WebAppLauncher.Launch`
- `WebAppLauncher.Launch.Params` – if launching with params

Parameters:

- `webAppId` – ID of web app assigned by platform vendor
- **relaunchIfRunning:** `relaunchIfRunning` – If supported on target platform, web app will force relaunch if value true
- **success:** `success` – Optional `WebAppLaunchSuccessBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

- (void) **launchWebApp:(NSString *)webAppId params:(NSDictionary *)params relaunchIfRunning:(BOOL)relaunchIfRunning success:(WebAppLaunchSuccessBlock)success failure:(FailureBlock)failure**
Launch a web application on the TV.

This method requires pairing on webOS

Related capabilities:

- `WebAppLauncher.Launch`
- `WebAppLauncher.Launch.Params` – if launching with params

Parameters:

- `webAppId` – ID of web app assigned by platform vendor
- **params:** `params` – Dictionary of key/value strings. Not available on all target platforms
- **relaunchIfRunning:** `relaunchIfRunning` – If supported on target platform, web app will force relaunch if value true
- **success:** `success` – Optional `WebAppLaunchSuccessBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

- (void) **joinWebApp:(LaunchSession *)webAppLaunchSession success:(WebAppLaunchSuccessBlock)success failure:(FailureBlock)failure**
Join an active web app without launching/relaunching. If the app is not running/joinable, the failure block will be called immediately.

Related capabilities:

- `WebAppLauncher.Send`
- `WebAppLauncher.Receive`

Parameters:

- `webAppLaunchSession` – `LaunchSession` for the web app to be joined
- **success:** `success` – Optional `WebAppLaunchSuccessBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

- (void) **joinWebAppWithId:(NSString *)webAppId success:(WebAppLaunchSuccessBlock)success failure:(FailureBlock)failure**
Join an active web app without launching/relaunching. If the app is not running/joinable, the failure block will be called immediately.

Related capabilities:

- `WebAppLauncher.Send`
- `WebAppLauncher.Receive`

Parameters:

- `webAppId` – Unique identifier for the web app to be joined
- **success:** `success` – Optional `WebAppLaunchSuccessBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

- (void) **closeWebApp:(*LaunchSession* *)launchSession success:(**SuccessBlock**)success failure:(**FailureBlock**)failure**
Closes a web app with the provided `LaunchSession`.

Related capabilities:

- `WebAppLauncher.Close`

Parameters:

- `launchSession` – `LaunchSession` associated with the web app to be closed
- **success:** `success` – Optional `SuccessBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

- (void) **pinWebApp:(*NSString* *)webAppId success:(**SuccessBlock**)success failure:(**FailureBlock**)failure**

Parameters:

- `webAppId`
- **success:** `success` – Optional `SuccessBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

- (void) **unPinWebApp:(*NSString* *)webAppId success:(**SuccessBlock**)success failure:(**FailureBlock**)failure**

Parameters:

- `webAppId`
- **success:** `success` – Optional `SuccessBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

- (void) **isWebAppPinned:(*NSString* *)webAppId success:(**WebAppPinStatusBlock**)success failure:(**FailureBlock**)failure**

Parameters:

- `webAppId`
- **success:** `success` – Optional `WebAppPinStatusBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

- (*ServiceSubscription* *) **subscribeIsWebAppPinned:(*NSString* *)webAppId success:(**WebAppPinStatusBlock**)success failure:(**FailureBlock**)failure**

Parameters:

- `webAppId`
- **success:** `success` – Optional `WebAppPinStatusBlock` to be called on success
- **failure:** `failure` – Optional `FailureBlock` to be called on failure

Typedefs

WebAppLaunchSuccessBlock

`void(^)(WebAppSession *webAppSession)`

Success block that is called upon successful launch of a web app.

- `webAppSession`

Object containing important information about the web app's session. This object is required to perform many functions with the web app, including app-to-app communication, media playback, closing, etc.

ScreenMirroringControl

The `ScreenMirroringControl` capability protocol serves to define the methods required for displaying the mobile app screen to LG TV.

Methods

- (`id<ScreenMirroringControl>`) **ScreenMirroringControl**

- (`CapabilityPriorityLevel`) **screenMirroringControlPriority**

- (**void**) **startScreenMirroring** Requests to start the screen mirroring

- (**void**) **startScreenMirroringWithSettings:(nullable NSDictionary<NSString, id> *) *settings** Requests to start the screen mirroring after setting up.

Parameters:

- `settings` – screen mirroring settings

- (**void**) **pushSampleBuffer:(CMSampleBufferRef)sampleBuffer with:(RPSampleBufferType)sampleBufferType** Delivers video/audio data captured by Upload Extension to screen mirroring.

Parameters:

- `sampleBuffer` – A reference to an immutable sample buffer object
- **with:** `sampleBufferType` – The type of sample buffered

- (**void**) **stopScreenMirroring** Requests to stop the screen mirroring

- (**void**) **setScreenMirroringDelegate:(__weak id<ScreenMirroringControlDelegate>)delegate** Registers a delegate to receive events while running the screen mirroring.

Parameters:

- `delegate`

ScreenMirroringControlDelegate

`ScreenMirroringControlDelegate` allows your app to receive screen mirroring status information.

Methods

- (void) **screenMirroringDidStart:(BOOL) result** Calls to pass the result of a screen mirroring start request.

Parameters:

- result – Screen mirroring start result

- (void) **screenMirroringDidStop:(BOOL)result** Calls to pass the result of a screen mirroring stop request.

Parameters:

- result – Screen mirroring stop result

- (void) **screenMirroringErrorDidOccur:(ScreenMirroringError)error** Calls when an error occurs after starting the screen mirroring. For error types, refer to *ScreenMirroringError*.

Parameters:

- error – Screen mirroring error

RemoteCameraControl

The RemoteCameraControl capability protocol serves to define the methods required for using the mobile camera for the LG TV.

Methods

- (id<*RemoteCameraControl*>) **remoteCameraControl**

- (*CapabilityPriorityLevel*) **remoteCameraControlPriority**

- (UIView *) **startRemoteCamera** Requests to start the remote camera.

- Default Camera Settings: Front
- Default Sound Settings: With Sound

Returns:

- UIView - Returns an object for the UIView created to show the camera preview.

- (UIView *) **startRemoteCameraWithSettings:(nullable NSDictionary<NSString *, id> *) settings** Requests to start the remote camera after setting up the camera.

- kRemoteCameraSettingsMicMute: Mute setting
- kRemoteCameraSettingsLensFacing: Front/rear camera settings

Parameters:

- settings – Camera settings

Returns:

- UIView - Returns an object for the UIView created to show the camera preview.

- (void) **stopRemoteCamera** Requests to stop the remote camera

- (void) **setLensFacing:(int)lensFacing** Sets the front/rear camera lens use.

- Front camera settings: RemoteCameraLensFacingFront (Default)
- Rear camera settings: RemoteCameraLensFacingBack

Parameters:

- lensFacing – Camera lens direction

- (void) **setMicMute:(BOOL)micMute** Sets the mute function of the microphone. (Default: NO)

Parameters:

- micMute – Microphone mute settings

- (void) **setRemoteCameraDelegate:(__weak id<RemoteCameraControlDelegate>)delegate** Registers a delegate to receive events while running the remote camera.

Parameters:

- delegate

RemoteCameraControlDelegate

RemoteCameraControlDelegate allows your app to receive remote camera status information.

Methods

- (void) **remoteCameraDidPair** Calls when the remote camera and TV are first connected (You have to guide the user to accept the connection on the TV.)

- (void) **remoteCameraDidStart:(BOOL)result** Calls to pass success or failure of connection with TV after starting remote camera function

Parameters:

- result – Connection result with TV

- (void) **remoteCameraDidStop:(BOOL)result** Calls to pass the result of a remote camera stop request.

Parameters:

- result – Remote camera stop result

- (void) **remoteCameraDidPlay** Calls when data transmission starts by requesting remote camera execution from TV.

- (void) **remoteCameraDidChange:(RemoteCameraProperty)property** Calls when a camera setting is changed by TV App request. For the property types, refer to *RemoteCameraProperty*.

Parameters:

- property – Remote camera property

- (void) **remoteCameraErrorDidOccur:(RemoteCameraError)error** Calls when an error occurs after starting the remote camera. For error types, refer to *RemoteCameraError*.

Parameters:

- error – Remote camera error

5.16.5 Sessions

LaunchSession

Any time anything is launched onto a first screen device, there will be important session information that needs to be tracked. LaunchSession will track this data, and must be retained to perform certain actions within the session.

Properties

NSString * appId System-specific, unique ID of the app (ex. youtube.leanback.v4, 0000134, hulu)

NSString * name User-friendly name of the app (ex. YouTube, Browser, Hulu)

NSString * sessionId Unique ID for the session (only provided by certain protocols)

id rawData Raw data from the first screen device about the session. In most cases, this is an NSDictionary.

LaunchSessionType sessionType When closing a LaunchSession, the DeviceService relies on the sessionType to determine the method of closing the session.

DeviceService * service DeviceService responsible for launching the session.

Methods

- (BOOL) **isEqual:(LaunchSession *)launchSession** Compares two LaunchSession objects.

Parameters:

- launchSession – LaunchSession object to compare.

Returns: YES if both LaunchSession id and sessionId values are equal

- (void) **closeWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Closes the session on the first screen device. Depending on the sessionType, the associated service will have different ways of handling the close functionality.

Parameters:

- success – (optional) SuccessBlock to be called on success
- failure: failure – (optional) FailureBlock to be called on failure

+ (LaunchSession *) **launchSessionForAppId:(NSString *)appId** Instantiates a LaunchSession object for a given app ID.

Parameters:

- appId – System-specific, unique ID of the app

+ (LaunchSession *) **launchSessionFromJSONObject:(NSDictionary *)json** Deserializes a LaunchSession object from json object.

Parameters:

- json – Serialized LaunchSession object by `-[LaunchSession toJSONObject]`.

LaunchSessionType

LaunchSession type is used to help DeviceService's know how to close a LunchSession.

Properties

LaunchSessionTypeUnknown Unknown LaunchSession type, may be unable to close this launch session

LaunchSessionTypeApp LaunchSession represents a launched app

LaunchSessionTypeExternalInputPicker LaunchSession represents an external input picker that was launched

LaunchSessionTypeMedia LaunchSession represents a media app

LaunchSessionTypeWebApp LaunchSession represents a web app

WebAppSession

Overview

When a web app is launched on a first screen device, there are certain tasks that can be performed with that web app. WebAppSession serves as a second screen reference of the web app that was launched. It behaves similarly to LaunchSession, but is not nearly as static.

In Depth

On top of maintaining session information (contained in the launchSession property), WebAppSession provides access to a number of capabilities.

- MediaPlayer
- MediaControl
- Bi-directional communication with web app

MediaPlayer and MediaControl are provided to allow for the most common first screen use cases a media player (audio, video, & images).

The Connect SDK JavaScript Bridge has been produced to provide normalized support for these capabilities across protocols (Chromecast, webOS, etc).

Properties

LaunchSession * launchSession LaunchSession object containing key session information. Much of this information is required for web app messaging & closing the web app.

DeviceService * service DeviceService that was responsible for launching this web app.

id<WebAppSessionDelegate> delegate When messages are received from a web app, they are parsed into the appropriate object type (string vs JSON/NSDictionary) and routed to the WebAppSessionDelegate.

Methods

- **(instancetype) initWithLaunchSession:(LaunchSession *)launchSession service:(DeviceService *)service**
Instantiates a WebAppSession object with all the information necessary to interact with a web app.

Parameters:

- **launchSession** – LaunchSession containing info about the web app session
- **service**: service – DeviceService that was responsible for launching this web app

- (*ServiceSubscription* *) **subscribeWebAppStatus:(WebAppStatusBlock)success failure:(FailureBlock)failure**
Subscribes to changes in the web app's status.

Parameters:

- **success** – (optional) WebAppStatusBlock to be called on app status change
- **failure**: failure – (optional) FailureBlock to be called on failure

- (**void**) **joinWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Join an active web app without launching/relaunching. If the app is not running/joinable, the failure block will be called immediately.

Parameters:

- **success** – (optional) SuccessBlock to be called on join success
- **failure**: failure – (optional) FailureBlock to be called on failure

- (**void**) **closeWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Closes the web app on the first screen device.

Parameters:

- **success** – (optional) SuccessBlock to be called on success
- **failure**: failure – (optional) FailureBlock to be called on failure

- (**void**) **connectWithSuccess:(SuccessBlock)success failure:(FailureBlock)failure** Establishes a communication channel with the web app.

Parameters:

- **success** – (optional) SuccessBlock to be called on success
- **failure**: failure – (optional) FailureBlock to be called on failure

- (**void**) **disconnectFromWebApp** Closes any open communication channel with the web app.

- (**void**) **pinWebApp:(NSString *)webAppId success:(SuccessBlock)success failure:(FailureBlock)failure** Pin the web app on the launcher.

Parameters:

- **webAppId** – NSString webAppId to be pinned.
- **success**: success – Optional SuccessBlock to be called on success
- **failure**: failure – Optional FailureBlock to be called on failure

- (**void**) **unPinWebApp:(NSString *)webAppId success:(SuccessBlock)success failure:(FailureBlock)failure**
UnPin the web app on the launcher.

Parameters:

- **webAppId** – NSString webAppId to be unpinned.
- **success**: success – Optional SuccessBlock to be called on success
- **failure**: failure – Optional FailureBlock to be called on failure

- (**void**) **isWebAppPinned:(NSString *)webAppId success:(WebAppPinStatusBlock)success failure:(FailureBlock)failure**
To check if the web app is pinned or not

Parameters:

- **webAppId**
- **success**: success – Optional WebAppPinStatusBlock to be called on success

- **failure:** failure – Optional FailureBlock to be called on failure

- (void) **sendText:(NSString *)message success:(SuccessBlock)success failure:(FailureBlock)failure** Sends a simple string to the web app. The Connect SDK JavaScript Bridge will receive this message and hand it off as a string object.

Parameters:

- message
- **success:** success – (optional) SuccessBlock to be called on success
- **failure:** failure – (optional) FailureBlock to be called on failure

- (void) **sendJSON:(NSDictionary *)message success:(SuccessBlock)success failure:(FailureBlock)failure** Sends a JSON object to the web app. The Connect SDK JavaScript Bridge will receive this message and hand it off as a JavaScript object.

Parameters:

- message
- **success:** success – (optional) SuccessBlock to be called on success
- **failure:** failure – (optional) FailureBlock to be called on failure

Typedefs

WebAppStatusBlock

void(^)(WebAppStatus status)

Success block that is called upon successfully getting a web app's status.

- status
The current running & foreground status of the web app

WebAppPinStatusBlock

void(^)(BOOL status)

Success block that is called upon successfully getting a web app's status.

- status
The current running & foreground status of the web app

WebAppSessionDelegate

WebAppSessionDelegate provides callback methods for receiving messages from a running web app.

Methods

- (void) **webAppSession:(WebAppSession *)webAppSession didReceiveMessage:(id)message** This method is called when a message is received from a web app.

Parameters:

- `webAppSession` – `WebAppSession` that corresponds to the web app that sent the message
- **didReceiveMessage:** `message` – Message from the web app, either an `NSString` or a JSON object in the form of an `NSDictionary`

- (void) **webAppSessionDidDisconnect:(*WebAppSession* *)webAppSession** This method is called when a web app's communication channel (`WebSocket`, etc) has become disconnected.

Parameters:

- `webAppSession` – `WebAppSession` that became disconnected

WebAppStatus

Status of the web app

Properties

WebAppStatusUnknown Web app status is unknown

WebAppStatusOpen Web app is running and in the foreground

WebAppStatusBackground Web app is running and in the background

WebAppStatusForeground Web app is in the foreground but has not started running yet

WebAppStatusClosed Web app is not running and is not in the foreground or background

5.16.6 Info Objects

AppInfo

Normalized reference object for information about a `DeviceService`'s app. This object will, in most cases, be used to launch apps.

In some cases, all that is needed to launch an app is the app id. For these cases, a static constructor method has been provided.

Properties

NSString * id ID of the app on the first screen device. Format is different depending on the platform. (ex. `youtube.leanback.v4`, `0000001134`, `netflix`, etc).

NSString * name User-friendly name of the app (ex. `YouTube`, `Browser`, `Netflix`, etc).

id rawData Raw data from the first screen device about the app. In most cases, this is an `NSDictionary`.

Methods

- (BOOL) **isEqual:(*AppInfo* *)appInfo** Compares two `AppInfo` objects.

Parameters:

- `appInfo` – `AppInfo` object to compare.

Returns: YES if both `AppInfo` id values are equal

+ (*AppInfo* *) **appInfoForId:(NSString *)appId** Static constructor method.

Parameters:

- appId – ID of the app on the first screen device

ChannelInfo

Normalized reference object for information about a TV's channels. This object is required to set the channel on a TV.

Properties

NSString * id TV's unique ID for the channel

NSString * name User-friendly name of the channel

NSString * number TV channel's number (likely to be a combination of the major & minor numbers)

int majorNumber TV channel's major number

int minorNumber TV channel's minor number

id rawData Raw data from the first screen device about the channel. In most cases, this is an NSDictionary.

Methods

- (**BOOL**) **isEqual:(ChannelInfo *)channelInfo** Compares two ChannelInfo objects.

Parameters:

- channelInfo – ChannelInfo object to compare.

Returns: YES if both ChannelInfo number & name values are equal

ExternalInputInfo

Normalized reference object for information about a DeviceService's external inputs. This object is required to set a DeviceService's external input.

Properties

NSString * id ID of the external input on the first screen device.

NSString * name User-friendly name of the external input (ex. AV, HDMI1, etc).

BOOL connected Whether the DeviceService is currently connected to this external input.

NSURL * iconURL URL to an icon representing this external input.

id rawData Raw data from the first screen device about the external input. In most cases, this is an NSDictionary.

Methods

- (BOOL) **isEqual:(*ExternalInputInfo* *)externalInputInfo** Compares two ExternalInputInfo objects.

Parameters:

- externalInputInfo – ExternalInputInfo object to compare.

Returns: YES if both ExternalInputInfo id & name values are equal

ImageInfo

Normalized reference object for information about an image to be sent to a device through the MediaPlayer capability.

Properties

NSURL * url URL source of the image

***ImageType* type** Type of image (see ImageType enum)

NSInteger width Width of the image (optional)

NSInteger height Height of the image (optional)

Methods

- (**instancetype**) **initWithURL:(NSURL *)url type:(*ImageType*)type** Creates an instance of ImageInfo with given property values.

Parameters:

- url – URL source of the image
- **type:** type – Type of image (see ImageType enum)

Typedefs

ImageType

NSInteger

MediaControlPlayState

Properties

MediaControlPlayStateUnknown

MediaControlPlayStateIdle

MediaControlPlayStatePlaying

MediaControlPlayStatePaused

MediaControlPlayStateBuffering

MediaControlPlayStateFinished

MediaInfo

Normalized reference object for information about a media file to be sent to a device through the MediaPlayer capability. “Media file”, in this context, refers to an audio or video resource.

Properties

NSURL * url URL source of the media file

NSString * mimeType Mime-type of the media file

NSString * title Title of the media file (optional)

NSString * description Short description of the media file (optional)

NSTimeInterval duration Duration of the media file

NSArray * images Collection of ImageInfo objects to send, as necessary, to the device when launching media through the MediaPlayer capability.

SubtitleInfo * subtitleInfo Subtitle track for this media instance (optional).

Methods

- (**instancetype**) **initWithURL:(NSURL *)url mimeType:(NSString *)mimeType** Creates an instance of MediaInfo with given property values.

Parameters:

- url – URL source of the media file
- **mimeType:** mimeType – Mime-type of the media file

- (**void**) **addImage:(ImageInfo *)image** Adds an ImageInfo object to the array of images.

Parameters:

- image – ImageInfo object to be added

- (**void**) **addImages:(NSArray *)images** Adds an array of ImageInfo objects to the array of images.

Parameters:

- images - Array of ImageInfo objects to be added

MediaLaunchObject

MediaLaunchObject is a container object which holds LaunchSession object,MediaControl object/or and PlayList-Control object

Properties

id<MediaControl> mediaControl MediaControl object of Media player

id<PlayListControl> playListControl PlayList Control Object of Media player

LaunchSession * session Launch Session object of Media player

Methods

- (instancetype) initWithLaunchSession:(*LaunchSession* *)session andMediaControl:(id<*MediaControl*>)mediaControl
Creates an instance of MediaLaunchObject with given property values.

Parameters:

- session
- **andMediaControl:** mediaControl – MediaControl object used to control playback

- (instancetype) initWithLaunchSession:(*LaunchSession* *)session andMediaControl:(id<*MediaControl*>)mediaControl andPlayListControl:(id<*PlayListControl*>)playListControl

Parameters:

- session
- **andMediaControl:** mediaControl
- **andPlayListControl:** playListControl

ProgramInfo

Normalized reference object for information about a TV's program.

Properties

NSString * id ID of the program on the first screen device. Format is different depending on the platform.

NSString * name User-friendly name of the program (ex. Sesame Street, Cosmos, Game of Thrones, etc).

ChannelInfo * channelInfo Reference to the ChannelInfo object that this program is associated with

id rawData Raw data from the first screen device about the program. In most cases, this is an NSDictionary.

Methods

- (BOOL) isEqual:(*ProgramInfo* *)programInfo Compares two ProgramInfo objects.

Parameters:

- programInfo – ProgramInfo object to compare.

Returns: YES if both ProgramInfo id & name values are equal

SubtitleInfo

Represents a subtitle track used for media playing.

The URL is required, so the `-init` method will throw an exception. Please use the parameterized initializers.

This class is immutable.

Different services support specific subtitles formats:

- DLNA service supports SRT format only. Since there is no official specification for them, subtitles may not work on all DLNA-compatible devices.
- Netcast service supports SRT format only, through DLNA.

- Google Cast service supports WebVTT format only and has additional requirements: https://developers.google.com/cast/docs/ios_sender#cors-requirements
- FireTV service supports WebVTT format only. Subtitles on Fire TV are hidden by default and should be displayed manually by the user.
- WebOS service supports WebVTT format only. Server providing subtitles should support CORS headers, similarly to Cast service's requirements.

Properties

NSURL * url The subtitle track's URL.

NSString * mimeType The subtitle's mimeType.

NSString * language The subtitle's source language. The contents depend on the target device.

NSString * label A custom label that may be displayed by a device's media player.

Methods

+ (instancetype) **initWithURL:(NSURL *)url** Creates a new instance with the given url.

Parameters:

- url

+ (instancetype) **initWithURL:(NSURL *)url andBlock:(void (^)(SubtitleInfoBuilder *builder))block** Creates a new instance with the given url and properties set in the builder object.

Parameters:

- url
- **andBlock:** block

SubtitleInfoBuilder

Used to initialize a `SubtitleInfo` object in a convenient way. The properties are writable at this point, and then become readonly in a final object.

You should not create this object manually. It is passed as a parameter to `+[SubtitleInfo initWithURL:andBlock:]` method.

<http://www.annema.me/the-builder-pattern-in-objective-c>

Properties

NSString * mimeType The subtitle's mimeType.

NSString * language The subtitle's source language. The contents depend on the target device.

NSString * label A custom label that may be displayed by a device's media player.

TextInputStatusInfo

Normalized reference object for information about a text input event.

Properties

UIKeyboardType keyboardType Type of keyboard that should be displayed to the user.

BOOL isVisible Whether the keyboard is/should be visible to the user.

id rawData Raw data from the first screen device about the text input status. In most cases, this is an NSDictionary.

ScreenMirroringError

Enumerates error type

Properties

ScreenMirroringErrorGeneric The general error

ScreenMirroringErrorConnectionClosed The error that occurs when the network is disconnected

ScreenMirroringErrorDeviceShutdown The error that occurs when the TV shuts down

ScreenMirroringErrorRendererTerminated The error that occurs when the TV app is closed

RemoteCameraProperty

Enumerates property type

Properties

RemoteCameraLensFacingFront The front camera

RemoteCameraLensFacingBack The rear camera

RemoteCameraPropertyUnknown The unregistered attribute

RemoteCameraPropertyBrightness The brightness property

RemoteCameraPropertyWhitebalance The white balance property

RemoteCameraPropertyRotation The screen rotation properties

RemoteCameraError

Enumerates error type

Properties

RemoteCameraErrorGeneric The general error

RemoteCameraErrorConnectionClosed The error that occurs when the network is disconnected

RemoteCameraErrorDeviceShutdown The error that occurs when the TV shuts down

RemoteCameraErrorRendererTerminated The error that occurs when the TV app is closed

5.16.7 Advanced

ConnectableDeviceStore

ConnectableDeviceStore is a protocol which can be implemented to save key information about ConnectableDevices that have been connected to. Any class which implements this protocol can be used as DiscoveryManager's deviceStore.

A default implementation, DefaultConnectableDeviceStore, will be used by DiscoveryManager if no other ConnectableDeviceStore is provided to DiscoveryManager when startDiscovery is called.

Privacy Considerations

If you chose to implement ConnectableDeviceStore, it is important to keep your users' privacy in mind.

- There should be UI elements in your app to
 - completely disable ConnectableDeviceStore
 - purge all data from ConnectableDeviceStore (removeAll)
- Your ConnectableDeviceStore implementation should
 - avoid tracking too much data (indefinitely storing all discovered devices)
 - periodically remove ConnectableDevices from the ConnectableDeviceStore if they haven't been used/connected in X amount of time

Properties

NSDictionary * storedDevices A dictionary containing information about all ConnectableDevices in the ConnectableDeviceStore. To get a strongly-typed ConnectableDevice object, use the `getDeviceForUUID:` method.

Methods

- **(void) addDevice:(ConnectableDevice *)device** Add a ConnectableDevice to the ConnectableDeviceStore. If the ConnectableDevice is already stored, it's record will be updated.

Parameters:

- device – ConnectableDevice to add to the ConnectableDeviceStore

- **(void) updateDevice:(ConnectableDevice *)device** Updates a ConnectableDevice's record in the ConnectableDeviceStore. If the ConnectableDevice is not in the store, this call will be ignored.

Parameters:

- device – ConnectableDevice to update in the ConnectableDeviceStore

- **(void) removeDevice:(ConnectableDevice *)device** Removes a ConnectableDevice's record from the ConnectableDeviceStore.

Parameters:

- device – ConnectableDevice to remove from the ConnectableDeviceStore

- (*ConnectableDevice* *) **deviceForId:(NSString *)id** Gets a *ConnectableDevice* object for a provided id. The id may be for the *ConnectableDevice* object or any of the device's *DeviceServices*.

Parameters:

- id – Unique ID for a *ConnectableDevice* or any of its *DeviceService* objects

Returns: *ConnectableDevice* object if a matching id was found, otherwise will return nil

- (*ServiceConfig* *) **serviceConfigForUUID:(NSString *)UUID** Gets a *ServiceConfig* object for a provided UUID. This is used by *DiscoveryManager* to retain crucial service information between sessions (pairing code, etc).

Parameters:

- UUID – Unique ID for the service

Returns: *ServiceConfig* object if a matching UUID was found, otherwise will return nil

- (**void**) **removeAll** Clears out the *ConnectableDeviceStore*, removing all records.

DefaultConnectableDeviceStore

DefaultConnectableDeviceStore is an implementation of *ConnectableDeviceStore* provided by Connect SDK for your convenience. This class will be used by *DiscoveryManager* as the default *ConnectableDeviceStore* if no other *ConnectableDeviceStore* implementation is provided before calling *startDiscovery*.

Privacy Considerations

As outlined in *ConnectableDeviceStore*, this class takes the following steps to ensure users' privacy.

- Only *ConnectableDevices* that have been connected to will be permanently stored
- On load & store, *ConnectableDevices* that have not been discovered within the *maxStoreDuration* will be removed from the *ConnectableDeviceStore*

File Format

DefaultConnectableDeviceStore stores data in a JSON file named *Connect_SDK_Device_Store.json* in the documents directory.

Properties

double maxStoreDuration Max length of time for a *ConnectableDevice* to remain in the *ConnectableDeviceStore* without being discovered. Default is 3 days, and modifications to this value will trigger a scan for old devices. *ConnectableDevices* that have been connected to will never be removed from the device store unless *remove:* or *removeAll* are called.

double created Date (in seconds from 1970) that the *ConnectableDeviceStore* was created.

double updated Date (in seconds from 1970) that the *ConnectableDeviceStore* was last updated.

int version Current version of the *ConnectableDeviceStore*, may be necessary for migrations

5.16.8 Globals

ConnectStatusCode

Helpful status codes that augment the localizedDescriptions of NSErrors that crop up throughout many places of the SDK. Most NSErrors that Connect SDK provides will have a ConnectStatusCode.

Properties

ConnectStatusCodeError Generic error, unknown cause

ConnectStatusCodeTvError The TV experienced an error

ConnectStatusCodeCertificateError SSL certificate error

ConnectStatusCodeSocketError Error with WebSocket connection

ConnectStatusCodeNotSupported Requested action is not supported

ConnectStatusCodeArgumentError There was a problem with the provided arguments, see error description for details

ConnectStatusCodeNotConnected Device is not connected

Globals

Typedefs

GCDWebServerAsyncStreamBlock

```
void(^)(GCDWebServerBodyReaderCompletionBlock completionBlock)
```

The GCDWebServerAsyncStreamBlock works like the GCDWebServerStreamBlock except the streamed data can be returned at a later time allowing for truly asynchronous generation of the data.

The block must call “completionBlock” passing the new chunk of data when ready, an empty NSData when done, or nil on error and pass a NSError.

The block cannot call “completionBlock” more than once per invocation.

GCDWebServerBodyReaderCompletionBlock

```
void(^)(NSData *data, NSError *error)
```

The GCDWebServerBodyReaderCompletionBlock is passed by GCDWebServer to the GCDWebServerBodyReader object when reading data from it asynchronously.

GCDWebServerMatchBlock

```
)(NSString *requestMethod, NSURL *requestURL, NSDictionary *requestHeaders, NSString *urlPath, NSDictionary *urlQuery)
```

The GCDWebServerMatchBlock is called for every handler added to the GCDWebServer whenever a new HTTP request has started (i.e. HTTP headers have been received). The block is passed the basic info for the request (HTTP method, URL, headers...) and must decide if it wants to handle it or not.

If the handler can handle the request, the block must return a new `GCDWebServerRequest` instance created with the same basic info. Otherwise, it simply returns `nil`.

GCDWebServerCompletionBlock

```
void(^)(GCDWebServerResponse *response)
```

The `GCDWebServerAsynchronousProcessBlock` works like the `GCDWebServerProcessBlock` except the `GCDWebServerResponse` can be returned to the server at a later time allowing for asynchronous generation of the response.

The block must eventually call “completionBlock” passing a `GCDWebServerResponse` or `nil` on error, which will result in a 500 HTTP status code returned to the client. It’s however recommended to return a `GCDWebServerErrorResponse` on error so more useful information can be returned to the client.

GCDWebServerProcessBlock

```
(GCDWebServerRequest *request)
```

The `GCDWebServerProcessBlock` is called after the HTTP request has been fully received (i.e. the entire HTTP body has been read). The block is passed the `GCDWebServerRequest` created at the previous step by the `GCDWebServerMatchBlock`.

The block must return a `GCDWebServerResponse` or `nil` on error, which will result in a 500 HTTP status code returned to the client. It’s however recommended to return a `GCDWebServerErrorResponse` on error so more useful information can be returned to the client.

GCDWebServerStreamBlock

```
NSData *(^)(NSError **error)
```

The `GCDWebServerStreamBlock` is called to stream the data for the HTTP body. The block must return either a chunk of data, an empty `NSData` when done, or `nil` on error and set the “error” argument which is guaranteed to be non-NULL.

FailureBlock

```
void(^)(NSError *error)
```

Generic asynchronous operation response error handler block. In all cases, you will get a valid `NSError` object. Connect SDK will make all attempts to give you the lowest-level error possible. In cases where an error is generated by Connect SDK, an enumerated error code (`ConnectStatusCode`) will be present on the `NSError` object.

Low-level error example

Situation

Connect SDK receives invalid XML from a device, generating a parsing error

Result

Connect SDK will call the `FailureBlock` and pass off the `NSError` generated during parsing of the XML.

High-level error example

Situation

An invalid value is passed to a device capability method

Result

The capability method will immediately invoke the `FailureBlock` and pass off an `NSError` object with a status code of `ConnectStatusCodeArgumentError`.

- error

`NSError` object describing the nature of the problem. Error descriptions are not localized and mostly intended for developer use. It is not recommended to display most error descriptions in UI elements.

SuccessBlock

`void(^)(id responseObject)`

Generic asynchronous operation response success handler block. If there is any response data to be processed, it will be provided via the `responseObject` parameter.

- responseObject

Contains the output data as a generic object reference. This value may be any of a number of types (`NSString`, `NSDictionary`, `NSArray`, etc). It is also possible that `responseObject` will be `nil` for operations that don't require data to be returned (move mouse, send key code, etc).

5.16.9 Misc

AppStateChangeNotifier

Listens to app state change events (`DidEnterBackground` and `DidBecomeActive`, in particular) and allows other components be notified about them using a simpler API.

Properties

AppStateChangeBlock **didBackgroundBlock** The block is called when the app has entered background.

AppStateChangeBlock **didForegroundBlock** The block is called when the app has entered foreground.

id<BlockRunner> blockRunner The `BlockRunner` instance specifying where to run the blocks. The default value is the main dispatch queue runner. Cannot be `nil`, as it will reset to the default value.

Methods

- (void) **startListening** Starts listening for app state change events. This method is idempotent.

You **MUST** call `-stopListening` for this object to be removed.

- (void) **stopListening** Stops listening for app state change events. This method is idempotent.

This method **MUST** be called to `dealloc` this object if you called `-startListening` before.

Typedefs

AppStateChangeBlock

```
void(^)()
```

Type of a block that is called on an app state change event.

BlockRunner

Abstracts and encapsulates asynchrony, that is how and where blocks are run. Using this protocol, you can easily change which dispatch queue or `NSOperationQueue` delegate blocks are run on, instead of hard-coding `dispatch_async(dispatch_get_main_queue(), ^{ })`. For example:

```
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND,
↳ 0);
AppStateChangeNotifier *notifier = [AppStateChangeNotifier new];
notifier.blockRunner = [[DispatchQueueBlockRunner alloc] initWithDispatchQueue:queue];
```

Another great use case is turning asynchronous tests into synchronous, making them faster and easier:

```
- (void)testStartListeningShouldSubscribeToDidEnterBackgroundEvent {
    AppStateChangeNotifier *notifier = [AppStateChangeNotifier new];
    notifier.blockRunner = [SynchronousBlockRunner new];
    [notifier startListening];

    __block BOOL verified = NO;
    notifier.didBackgroundBlock = ^{
        verified = YES;
    };
    [self postNotificationName:UIApplicationDidEnterBackgroundNotification];

    XCTAssertTrue(verified, @"didBackgroundBlock should be called");
}
```

Here we use the synchronous block runner (instead of the default asynchronous, main queue one) to avoid writing asynchronous tests with `XCTestExpectation`.

Methods

- (void) **runBlock:(nonnull VoidBlock)block** Runs the given `block` somewhere, depending on the concrete implementation.

Parameters:

- `block` – block to run; must not be `nil`.

- (void) **runBlock:(nonnull VoidBlock)block** Runs the given `block` somewhere, depending on the concrete implementation.

Parameters:

- `block` – block to run; must not be `nil`.

Typedefs

VoidBlock

`void(^)(void)`

A type for blocks without arguments and no return value.

DispatchQueueBlockRunner

Dispatches a `block` asynchronously on the given `dispatch_queue_t` queue.

Please use the `-initWithDispatchQueue: initializer`, because you must specify the queue.

Methods

- (instancetype) **initWithDispatchQueue:(dispatch_queue_t)queue** Designated initializer. Initializes the object with the given `dispatch_queue` which will run the blocks. The `queue` must not be `nil`.

Parameters:

- `queue`

+ (instancetype) **mainQueueRunner** Convenience method that returns a block runner with the main dispatch queue.

SubscriptionDeduplicator

Deduplicates subscription notifications with the same state. The state can be of any class, allowing `NSNumber`-wrapped values.

It's an immutable class.

Methods

- (instancetype) **runBlock:(dispatch_block_t)block ifStateDidChangeTo:(id)newState** If the new `state` is different from the previous one, runs the `block` synchronously.

Parameters:

- `block`
- **ifStateDidChangeTo:** `newState`

Returns: a new instance that you should save to track the new state.

SynchronousBlockRunner

Runs a `block` synchronously on the current thread/queue (that is, in the middle of `-runBlock:` call).

5.17 TV Web Apps

TV web apps are similar to standard web apps that use common web technologies such as HTML5, Javascript, and CSS. TV web apps are typically optimized for larger displays.

To learn more

1. Read the *Overview* article
2. Learn how easy it is to *Create a TV Web App*
3. Learn how to *Port a Receiver App to webOS*

5.17.1 Overview

What are TV web apps?

- Most TV apps are web apps that are packaged to run on the TV. They are developed using standard web technologies.
- TV web apps can be viewed on a TV without a browser, since they execute inside a web runtime environment.

Why TV web apps?

webOS TV, Chromecast, and Apple TV allow synchronized experience across multiple devices through web sockets. This enables users to interact with a TV web app using their mobile devices.

For example, if you created a TV chess board game app, users would not only interact with the app on the TV, they would also be able to interact with the app using their mobile devices.

Web app IDs

Mobile web apps require a web app id in order to launch on webOS TV and Chromecast. This web app id is translated into the mobile web app's URL when it is launched on the TV.

Platform	URL
Web app id for webOS TV	http://lgsvl.com/connectSDK/index.php
Web app id for Chromecast	https://developers.google.com/cast/docs/registration
Apple TV	Apple TV does not require a web app id.

Important: When designing your TV web app, be mindful of [OverScan](#). To avoid having parts of your web app cut off, we recommend not placing UI elements near the corner of the screen and always test your web apps to ensure they display properly on each targeted platform.

Interaction with TV web apps

All interactions with Chromecast and Apple TV web apps occur from a mobile device or laptop since the device does not support external remote controls. On other platforms such as webOS, the TV ships with traditional and [Magic Remotes](#). When designing your web app, make sure to design for the platforms you intend to support. On Chromecast and Apple TV, avoid using UI elements that make users think they are clickable. On webOS, make UI elements clickable since users may use their Magic Remote to interact with your web app.

Make sure to review all design guidelines for each platform you intend to support.

Web runtimes on various TV platforms may not be the same

While the HTML5 spec brings us one step closer to the “write once, run everywhere” utopia, we still recommend that you test your web app on each TV platform you intend to support.

- Web rendering engines vary which may cause inconsistency across platforms. For example, webOS uses WebKit 2.0 and it is not officially documented what Chromecast and others use.
- Hardware differences between dongles, set-top boxes, and Smart TVs can be significant - therefore, complex animations and computations should be reviewed.
- Screen resolution can vary between platforms and devices. webOS Smart TVs run at 1080P (1920x1080) while Chromecast currently renders WebView in 720P (1280x720). Apple TV automatically adjusts to match the resolution of the connected TV.
- Lastly, video and audio codec support can also cause fragmentation across multiple platforms.

Our experience has shown that using standard design patterns such as responsive design and standard video formats (MP4) - there is little variation between most platforms.

5.17.2 Create a TV Web App

Connecting a web app with the JavaScript bridge is incredibly simple and requires a minimum amount of effort. First, make sure you’ve got the right scripts imported.

- [Google Cast SDK JavaScript Receiver file](#)
- [Connect SDK JavaScript Bridge](#)

```
<script src="//www.gstatic.com/cast/sdk/libs/receiver/2.0.0/cast_receiver.js"
↪ language="JavaScript" type="text/javascript"></script>
<script src="connect_bridge.min.js" language="JavaScript" type="text/javascript"></
↪ script>
```

After scripts are imported, it is a simple matter to get your app configured. No matter what platform you are running on, the proper setup will occur to enable your web app.

```
window.connectManager = new connectsdk.ConnectManager();
window.connectManager.init();
```

Of course, if you actually want to enable any functionality in your web app, you will have to do a little more work. Integration with Connect SDK happens on two different levels.

Media playback and control

```
window.mediaElement = document.getElementById('media');
window.connectManager.setMediaElement(window.mediaElement);
```

Bi-directional communication

Receiving messages

```

window.connectManager.on("message", function(data) {
    console.log("Got message from sender " + data.from);
    console.log("Got message from mobile device " + data.message);
});

```

Sending messages

```

window.connectManager.sendMessage(to, "This is a test message");
window.connectManager.sendMessage(to, { "message" : "This is a JSON test message" });

window.connectManager.broadcastMessage("This is a test message");
window.connectManager.broadcastMessage({ "message" : "This is a JSON test message" });

```

5.17.3 Port a Receiver App to webOS

The Connect SDK JavaScript Bridge has been designed to enable near feature-parity with existing platforms. Ideally, one web app should be capable of running across the range of TV platforms available on the market.

This article will take a Custom “Receiver” developed for Chromecast and port it to work on both webOS and Chromecast through Connect SDK.

Here is the code for the Chromecast-specific app.

```

<!doctype html>
<html>
<head>
    <title>Chromecast Custom Receiver</title>
</head>
<body>
    <video id='media' />
    <script src="//www.gstatic.com/cast/sdk/libs/receiver/2.0.0/cast_receiver.js"></
    ↪script>
    <script>
        window.onload = function() {
            window.mediaElement = document.getElementById('media');
            window.mediaManager = new cast.receiver.MediaManager(window.mediaElement);

            window.castReceiverManager = cast.receiver.CastReceiverManager.
            ↪getInstance();

            window.castMessageBus = window.castReceiverManager.getCastMessageBus (
            ↪"urn:x-cast:com.example.MyApp");
            window.castMessageBus.addEventListener("message", function(message) {
                window.castMessageBus.broadcast("Got your message");
            });

            window.castReceiverManager.start();
        };
    </script>
</body>
</html>

```

There are a few things happening here.

1. The Chromecast SDK is being loaded
2. On page load, Chromecast SDK is being initialized
3. While initializing Chromecast, it is given a reference to our media element
4. A channel for communication is being established with a response on each message received
5. Event listeners are being added to the media element to track play state

With the Connect SDK JavaScript Bridge, these steps remain very similar.

1. This Chromecast SDK is being loaded
2. The Connect SDK JavaScript Bridge is being loaded
3. On page load, Connect SDK is being initialized
4. While initializing Connect SDK, it is given a reference to our media element
5. A channel for communication is being established with a response on each message received
6. Event listeners are being added to the media element to track play state

See the Connect SDK implementation below.

```
<!doctype html>
<html>
<head>
  <title>Connect SDK Web App</title>
</head>
<body>
  <video id='media' />
  <script src="//www.gstatic.com/cast/sdk/libs/receiver/2.0.0/cast_receiver.js"></
  <script>
  <script src="connectsdk.js"></script>
  <script>
    window.onload = function() {
      window.connectManager = new connectsdk.ConnectManager();

      window.mediaElement = document.getElementById('media');
      window.connectManager.setMediaElement(window.mediaElement);

      window.connectManager.on("message", function(data) {
        window.connectManager.sendMessage(data.from, "Got your message");
      });

      window.connectManager.init();
    };
  </script>
</body>
</html>
```

In this basic example, we were able to port an app from one platform to two by only adding one line of code (a JavaScript file import). Under the hood, the Connect SDK JavaScript bridge will run the initialization for whichever platform it is detected as running on.

We encourage you to attach media events directly to your media element to avoid having to add platform-specific code to your web app.

Portions of this page are modifications based on work created and shared by Google and used according to terms described in the [Creative Commons 3.0 Attribution License](#).

5.18 Release

5.18.1 ConnectSDK v1.6.0 Released

Deepak Sharma | Posted by Deepak Sharma | September 9, 2015

We proudly announce the launch of ConnectSDK version 1.6.

New in this release, you can get your app to work with Android TVs. Isn't it cool?

To improve the playback experience, we have added the support for subtitles. And great news for Cordova developers; this version makes the build process really simple, with better playback experience with support for subtitles and playlist controls and much more.

Here is a list of what's new the ConnectSDK version *1.6.0* offers:

- Cordova support
 - Automatic install scripts for iOS and Android
 - Support for pinning web apps
 - Support for subtitles
 - Support for pairing type
 - Support for playlist controls
 - API for external input picker
 - Simplified way to determine device capabilities
 - Miscellaneous Bug fixes
- Subtitle support on WebOS, Netcast, DLNA, Chrome cast and FireTV.
- Support for Android TV devices.
- Fixed play media issue on Roku 6.2
- Removed Rewind and FastForward capabilities from Netcast service
- Miscellaneous bug fixes.

Please continue to help our work by contributing to our open-source effort and providing your valuable *feedback to us*.

5.18.2 ConnectSDK v1.5 Announcement

Alpesh Saraiya | Posted by Alpesh Saraiya | July 9, 2015

With ConnectSDK version 1.5, we're extremely proud to announce support for Amazon's Fling SDK on Fire TV and Fire Stick devices! iOS and Android Apps can now seamlessly beam video, audio, and images to Fire TV / Fire Stick. We are looking forward to expanding support for Amazon products. To further integrate apps within Smart TVs, we now have ability to pin web apps on webOS 2014+ TVs. Developers can now provide convenient, instant access to a wealth of web apps like MusixMatch. If you have a web app to offer and want to increase your install base with minimal effort, please register [here](#). The combined iOS and Android release notes for ConnectSDK v1.5:

- Supports Amazon Fling SDK to play and control media on Fire TV devices
- Pinning web app on webOS TV launcher bar
- Enhanced webOS TV media player
 - Added playlist and loop support

- Extended play state subscription to handle media playback errors
- Added launching input picker for new versions of webOS TVs
- Fixed discovery for ChromeCast in Android
- Added ConnectSDK support for [Windows](#) on LG webOS and NetCast Smart TVs (big thanks to contributor [Sorin Serban!](#))
- Created a first set of integration and acceptance tests

We are working towards supporting some more exciting new features and device platforms in the coming months. Please continue to help our work by contributing to our open-source effort and providing your valuable *feedback to us*.

5.18.3 Connect SDK 1.4.4 released

|Alpesh Saraiya| Posted by Alpesh Saraiya | April 27, 2015

With version 1.4.4, we've added support for Google Cast SDK 2.6.0, which allows streaming of audio to Google Cast-enabled speakers, such as LG Music Flow speaker. Also we now support AirPlay pin mode for increased security. Lastly, we've fixed a number of issues related to DLNA and other miscellaneous bugs.

The combined release notes for iOS and Android:

- Added AirPlay pin mode support
- Added LG Music Flow speaker support (Google Cast for Audio and DLNA)
- Support for Google Cast SDK 2.6.0
- Misc DLNA fixes
 - DLNA subscription methods
- Allow to set pairing type for WebOS TVs
- Miscellaneous bug fixes
 - Replaced DefaultHttpClient with HttpURLConnection
 - Added a new exception class - NotSupportedServiceCommandError
 - Compiler and static analyzer warnings
 - Immediate disconnect if Apple TV has an IPv6 address only
 - Lint warning

We are working towards some exciting features, including Windows SDK support and new device platforms support in the coming months. Please continue to help our work by providing your valuable *feedback to us*.

5.18.4 Xbox One & Sonos support added to 1.4.2 release

|Vivek Sekar| Posted by Vivek Sekar | February 10, 2015

With the release of the 1.4.2 version of Connect SDK we have added support for Xbox One and Sonos Speakers, bring the total number of platforms we support to 8. Along with the new platforms, we have added support for Playlist functionality and improved the SSDP classes also.

The combined release notes for iOS and Android:

- Support for Xbox One console and Sonos speakers
- Added playlist support over DLNA

- Fixed video playing on Roku firmware 6.1
- Significantly improved SSDP classes
- Added new API's to
 - Display image & Play media
- Fixed saving service configuration
- Added support for Android Studio 1.0
- API Integration tests
- Miscellaneous bug fixes

Continuing our focus on quality, we have added a new repository [Connect-SDK-Android-API-Sampler](#) that focuses on the testing the public API's that are available as part of the Android SDK. We will focus on the iOS SDK next.

We are working towards some really cool and interesting features in the upcoming releases and cannot wait to get it out. Support our work by providing your valuable *feedback to us*.

5.18.5 Connect SDK 1.4 is out

|Vivek Sekar| Posted by Vivek Sekar | December 3, 2014

We have just pushed out the code for the 1.4.0 release. Here is a quick overview of the additions we have made:

1.4.0 release notes

- Modularization of Connect SDK
- Improved support for DLNA devices
 - DLNA volume control subscriptions
 - DLNA play state subscriptions
 - DLNA media info
- Unit tests for the discovery services providers
- Bug fixes in iOS, Android and Google Cast modules

Modularization

With the growing adoption of Connect SDK, a frequently requested feature has been for the developers to be able to pick and choose the various devices they want to support in their applications. As they put it – it would allow them to only have the necessary components as part of their application - so apps that directly stream media content does not have to worry about the web app support or vice versa.

With the 1.4.0 release we are taking our first steps towards achieving modularization within the features offered by Connect SDK. The 1.4.0 release allows developers to be able to pick between

- **full** (all you can eat version)
- **lite** (Connect SDK without the Google Cast) versions of Connect SDK.

Going forward in the upcoming releases we will add more and more of the existing features into this modularized approach. So you can pick and choose the features, like DIAL, Google Cast, Roku, Apple TV, LG Smart TV's, DLNA.

DLNA

With over 18,000 device models supporting DLNA, we are putting our efforts to be able to address these plethora of devices. With the 1.4.0 release we have further improved the support for DLNA devices. With this release we have added Volume control, play state & media info subscription. Along with some bug fixes to improve stability.

Unit tests

As Connect SDK grows to support more and more platforms and their SDK's, We have started work towards having a better overview on the quality of the code we are pushing out and integrating with. With the 1.4.0 release, we have started adding unit test coverage for the search discovery providers. Going forward the work on the test coverage will continue independent of the Connect SDK's release cycle, so that we can catch up to all the work that has been put out till now.

Just like the previous releases, we look forward to your feedback. We have already started working on Connect SDK v1.4.1 and look forward to sharing it with you soon!

5.19 Article

5.19.1 Connect SDK Smart Home demo

|Vivek Sekar| Posted by Vivek Sekar | April 8, 2015

We have spent the last month working with some exciting Smart Home products & technologies. Now we are ready to showcase our work.

We believe the Smart Homes of the future, are not going to be driven by devices from a single manufacturer, instead a network of devices from various manufacturers. And interoperability will be a key part of the experience for these devices to be able to deliver on the promise of simplifying the user's life.

We hope to use the understanding from this showcase, to be able to deliver a solution for developers to be able to leverage the various SDK's out there to provide novel and innovative application solutions for the user.

We have made the Smart Home sampler app source code available in github.

This demo app demonstrates a scenario of using various Smart Home devices in two home scenes. They represent a living room and a family room, each containing a media device, light bulbs, and possibly other devices. The supported devices come from different categories (media players, light bulbs, switches, and iBeacons) and multiple manufacturers.

The scenario of the app is:

1. You enter the living room, which is detected by an iBeacon,
2. A playlist starts to play on a TV or speaker, and the light bulbs change color to match one of the colors of the album art during playback.
3. Then the user moves from the living room scene to the family room scene.
4. Where the session information is transfered from the living room to the family room.
 - The devices in the living switch off and the session is picked up in the family room
5. The user put the scene to sleep using voice command (to replicate control using Siri or Google Now or other voice engine/assitants)
 - The speaker fades out the music, while the LED bulb fade out and switch off.
6. The Scene wakes up after a defined time - to mimic waking up from an alarm.
 - The Led Bulbs switch on along with speaker.

For the source code & additional information

- <https://github.com/ConnectSDK/SmartHomeSamplerAndroid>

- <https://github.com/ConnectSDK/SmartHomeSampleriOS>

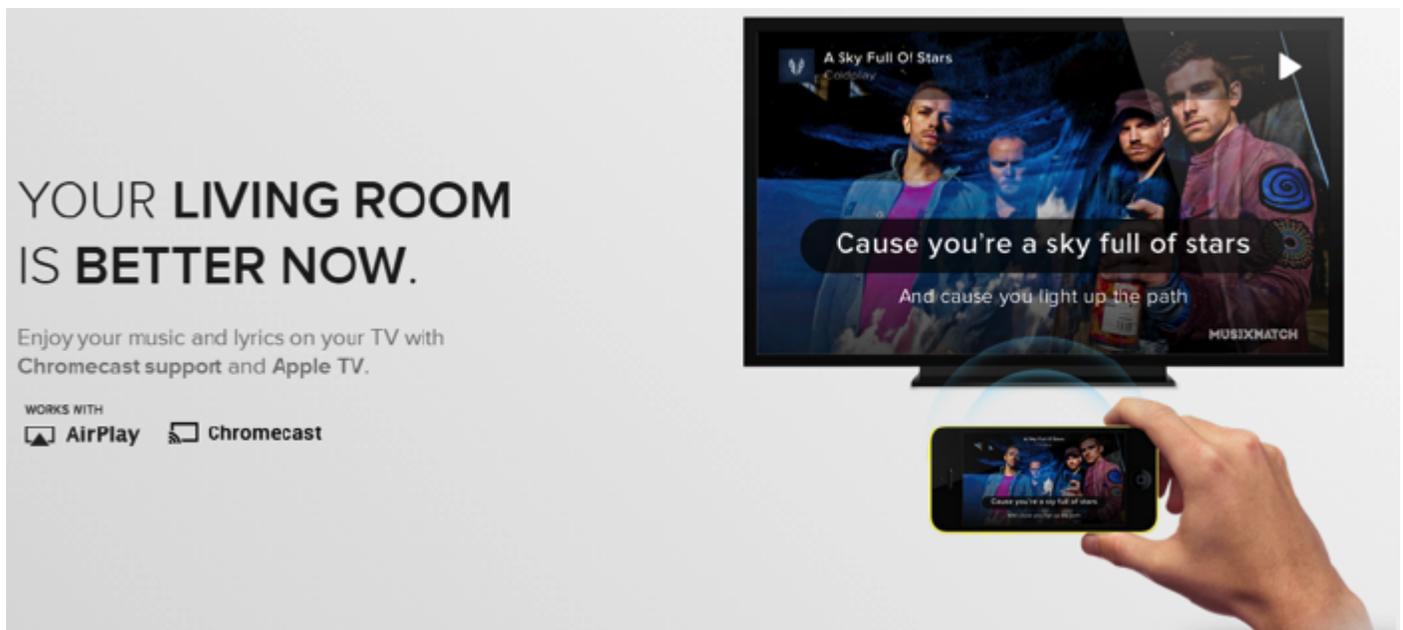
Support our work by providing your valuable *feedback to us*.

5.19.2 Recently Launched Connect SDK Apps and Upcoming 1.4 Release

|Chris Cukor| Posted by Chris Cukor | September 29, 2014

Developers are excited about Connect SDK because it solves a lot of their day to day problems. We wanted share a few recent examples of how Connect SDK is being used for music, premium content, and personal media.

MusiXMatch – Your favorite music with lyrics. Beam it all to your TV screen to enjoy with friends and family.



SnagFilms – The award-winning streaming video platform offers entertainment lovers an extensive library of over 5,000 free movies, TV series and web originals on demand.



Seagate Media App – If you back up your personal media to one of Seagate's enabled drives (Seagate Central, Wireless Plus or LaCie Fuel), now you can enjoy your pictures, movies and music on your TV.



We are working with more developer whose apps will be launching this year and we'll be sure to keep you posted on the highlights.

Check back soon for details about the upcoming 1.4 release that will support a host of new features and devices.

5.19.3 Connect SDK now supports Apple TV

|Henry Levak| Posted by Henry Levak | June 10, 2014

All of the devices we use should work harmoniously together - and in some cases they do. Take for example when you receive an email, you have the ability to read it on your mobile phone, tablet, or PC. Similarly, when you begin a Netflix movie on your desktop, you can finish watching it on your tablet and many other devices. Consumers are beginning to expect this connected experience between some of their devices - but few expect it from the biggest screen in their house. Being able start something on one device and continue it on the big screen is not as widely supported as it should be - and we want to play a part in changing that.

While many app developers acknowledge the opportunity a big, high definition display can bring (other than a few Chromecast-enabled apps) very few have implemented any app-to-TV functionality, and we don't blame them. The reality is, there are too many second screen protocols to choose from and the level of effort to integrate can be very high. Not to mention, the market share of each protocol individually makes it difficult to prioritize it over other opportunities.

We saw these roadblocks for app developers as a huge opportunity and so we designed and built Connect SDK. Our goal was simple, we wanted to expand the reach of second screen development by tackling the ever growing array of second screen protocols. Our result being, a single SDK with integrated support for multiple protocols, in which the effort of dealing with each one is abstracted away and the size of the opportunity is an aggregation of multiple platforms.

It wasn't too long ago in April 2014 that we launched Connect SDK with support for five TV platforms. Today we are excited to announce that Connect SDK supports Apple TV with the release of version 1.3.

What does this mean?

For the **Android app developer**, you can now beam photos, videos, and audio files to Apple TVs. By using an undocumented protocol, Connect SDK lets Android developers discover, connect to, and control Apple TVs, much like webOS and Roku devices (for a full list of supported features, see [Supported Features](#)). And, because Connect SDK abstracts all protocols, beaming a photo to an Apple TV is just as easy as beaming it to a Chromecast or LG Smart TV '13.

For the **iOS developer**, you can choose between two modes, “Mirrored” and “Media”.

- In **Mirrored mode**, web app beaming is accomplished by using AirPlay to mirror a secondary display that is actually being rendered on the iOS device. This allows developers to build full screen TV-optimized web applications that work across webOS, Chromecast, and now Apple TV. In order to use this mode, the user will need to enable AirPlay mirroring in the Control Center. Also, as with any Airplay mirroring app - TV experience will end if the user switches away from your app.
- In **Media mode**, photos, videos, and audio is beamed directly to an Apple TV. Using this mode provides the most seamless user experience, but before using it, please review Apple’s developer guidelines as it is enabled by an undocumented protocol. While all protocols are subject to change with software updates, undocumented protocols may be particularly so.

As with any release, we look forward to your feedback. We have already started working on Connect SDK v1.4 and look forward to sharing it with you soon!

5.20 Terms and Conditions

5.20.1 Copyright / Website Information

This website is owned and operated by LG Electronics Inc. (“LGE”). This website page may contain proprietary notices and copyright information, the terms of which must be observed and followed. Users of the site may download or print one copy of any and all materials on the site for personal, non-commercial use, provided that they do not modify or alter the materials in any way, nor delete or change any copyright or trademark notice. All material on this site is provided for lawful purposes only. None of the information on this site may be copied, distributed or transmitted in any way for commercial use without the express written consent of LGE. LGE reserves full ownership of and intellectual property rights in any materials downloaded from this site.

5.20.2 Submissions

LGE does not want to receive confidential or proprietary information from you through our website. Please note that any information or material sent to LGE will be deemed NOT to be confidential. By sending LGE any information or materials, you grant LGE an unrestricted, irrevocable license to use, reproduce, display, perform, modify, transmit and distribute those materials or information in any media now or hereinafter existing, and you also agree that LGE is free to use any ideas, concepts, know-how or techniques that you send us for any purpose.

5.20.3 Disclaimer of Warranty / Limitation of Liability

INFORMATION ON THIS WEBSITE IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OF NONINFRINGEMENT. SOME JURISDICTIONS DO NOT ALLOW FOR THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSIONS MAY NOT APPLY TO YOU. IN NO EVENT WILL LGE BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY INDIRECT, CONSEQUENTIAL, EXEMPLARY, INCIDENTAL, SPECIAL OR PUNITIVE DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFIT DAMAGES ARISING FROM YOUR USE OF THIS WEBSITE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. NOTWITHSTANDING ANYTHING

TO THE CONTRARY CONTAINED HEREIN, LGE'S LIABILITY TO YOU FOR ANY CAUSE WHATSOEVER AND REGARDLESS OF THE FORM OF THE ACTION, WILL AT ALL TIMES BE LIMITED TO \$100.00.

5.20.4 Third Party Websites

Certain links on this website link to third party websites outside the of LGE. LGE is not responsible for and disclaims all liability with respect the content of any linked site or any link contained in a linked site and makes no representations or warranties with respect to such third party sites.. The inclusion of any link in this website does not imply an endorsement by LGE of any third party site or the information contained therein.

5.20.5 General

This website may contain inaccuracies and typographical errors. LGE does not warrant the accuracy or completeness of the materials herein or the reliability of any advice, opinion, statement or other information displayed or distributed through this website. You acknowledge that any reliance on any such opinion, advice, statement, or information shall be at your sole risk. LGE reserves the right, in its sole discretion, to correct any errors or omissions in any portion of the site. LGE may make changes to this website, at any time without notice. This Agreement operates to the fullest extent permissible by law. If any provision of this Agreement is unlawful, void or unenforceable, that provision is deemed severable from this Agreement and does not affect the validity and enforceability of any remaining provisions.

LGE may from time to time amend these Terms and Conditions and additional terms that may apply to your use of this website, to the extent permitted under applicable laws and regulations.

Use of the LGE Service after the amended Terms of Use goes into effect will constitute your consent to such amendment. You may revoke your consent to these Terms of Use by terminating your Account at any time, upon which you will not be subject to the application of the amended Terms of Use.

5.20.6 Procedure for Resolving Dispute

Except to the extent prohibited by local law, any dispute arising out of or in connection with these Terms of Use, including any question regarding its existence, validity or termination, shall be referred to and finally resolved by arbitration (i) under the Rules of the Korean Commercial Arbitration Board (of which rules are deemed to be incorporated by reference into this clause), (ii) where the number of arbitrators shall be one, (iii) the seat, or legal place, of arbitration shall be Seoul, Republic of Korea, (iv) the language to be used in the arbitral proceedings shall be English and (v) the governing law of the contract shall be the substantive law of the Republic of Korea.

To the extent required by local law in order for the arbitration to be valid and legally effective as a means of dispute resolution, including as against a consumer, reference to the Rules of the Korean Commercial Arbitration Board in (i) above shall be deemed to refer to the rules of the most prominent arbitration body (the "Local Arbitration Rules") in your country, and reference to Seoul, Republic of Korea in (iii) above shall be deemed to refer to the capital city of your country.

You may only resolve disputes with us on an individual basis, and not as a plaintiff or class member in any purported class or representative proceedings.

If you have any questions about these terms of use, please contact us at developer@lge.com.

5.21 Cookie Policy

The connectsdk.com website (the "Site") use cookies. You can find out more about cookies and how to control them below.

By using the Sites, you accept the use of cookies in accordance with this cookie policy. If you do not accept the use of these cookies, please disable them following the instructions in this cookie policy.

5.21.1 What is a cookie?

Cookies are text files containing small amounts of information which are downloaded to your computer or mobile device when you visit a website. Cookies are then sent back to the originating website on each subsequent visit, or to another website that recognizes that cookie. Cookies are useful because they allow a website to recognize a user's device.

Cookies do lots of different jobs, like letting you navigate between pages efficiently, remembering your preferences, and generally improving the user experience. They can also help to ensure that adverts you see online are more relevant to you and your interests.

For further information on cookies, including how to see what cookies have been set on your device and how to manage and delete them, visit <http://www.allaboutcookies.org/>.

5.21.2 What cookies do we use on the Sites?

We use the following cookies.

Strictly necessary cookies. These are cookies that are required for the operation of our website. They include, for example, cookies that enable you to log into secure areas of our website. These cookies do not gather information about you that could be used for marketing or remembering where you have been on the internet. This category of cookies cannot be disabled.

Analytical cookies. They allow us to recognize and count the number of visitors and to see how visitors move around our website when they are using it. This helps us to improve the way our website works, for example, by ensuring that users are finding what they are looking for easily. These cookies do not collect information that can identify you. All the information that these cookies collect is anonymous and is only used to improve how the website works.

Our website uses **Google Analytics cookies.** Information collected by Google Analytics cookies will be transmitted to and stored by Google on servers in the United States of America in accordance with its privacy practices. To see an overview of privacy at Google and how this applies to Google Analytics, visit > <http://www.google.co.uk/intl/en/analytics/privacyoverview.html>. You may opt out of tracking by Google Analytics by visiting > <https://tools.google.com/dlpage/gaoptout?hl=en-GB>.

Advertising Cookies (Behavioral advertising). To personalize our Sites, deliver customized advertisements to you, or contact you directly where you have separately consented to such communications, in a way which is relevant to you and which matches your interests by, for example, using information about products you have browsed or ordered on our website.

Functional cookies. These are used to recognize you when you return to our website. This enables us to personalize our content for you and remember your preferences (for example, your choice of language or region). These cookies do not collect information that can identify you. All the information that these cookies collect is anonymous and is only used to improve how the website works.

You can find more information about the individual cookies we use and the purposes for which we use them in the table below:

Cookie Type	Cookie Name	Source	Expiration	Purpose
Strictly necessary	CookieScriptConsent	LG Electronics	2 years	This cookie is used by Cookie-Script.com service to remember visitor cookie consent preferences. It is necessary for Cookie-Script.com cookie banner to work properly.
Analytics	_ga	LG Electronics	2 years	This cookie name is associated with Google Analytics - which is a significant update to Google's more commonly used analytics service. This cookie is used to distinguish unique users by assigning a randomly generated number as a client identifier. It is included in each page request in a site and used to calculate visitor, session and campaign data for the sites analytics reports.
Analytics	_ga_L240ET5MQ8	LG Electronics	2 years	This cookie name is associated with Google Analytics - which is a significant update to Google's more commonly used analytics service. This cookie is used to distinguish unique users by assigning a randomly generated number as a client identifier. It is included in each page request in a site and used to calculate visitor, session and campaign data for the sites analytics reports.
Analytics	_gat_gtag_UA_17997319_1	LG Electronics	1 minute	This cookie is part of Google Analytics and is used to limit requests (throttle request rate).
Analytics	_gat_gtag_UA_17997319_5	LG Electronics	1 minute	This cookie is part of Google Analytics and is used to limit requests (throttle request rate).
Analytics	_gid	LG Electronics	1 day	This cookie is set by Google Analytics. It stores and updates a unique value for each page visited and is used to count and track pageviews.
Advertising	VISITOR_INFO1_LIVE	Google LLC YouTube	6 months	This cookie is set by YouTube to keep track of user preferences for YouTube videos embedded in sites. It can also determine whether the website visitor is using the new or old version of the YouTube interface.
Advertising	YSC	Google LLC YouTube	Session	This cookie is set by YouTube to track views of embedded videos.

5.21.3 How to refuse, disable or delete cookies?

You can refuse certain types of cookies (except “strictly necessary cookies”) at any time by changing your settings on Cookie Settings.

You may also disable cookies by activating the setting on your browser that allows you to refuse the setting of all or some cookies. However, if you use your browser settings to disable all cookies (including strictly necessary cookies) you may not be able to access all or parts of the Sites.

Disabling a cookie or category of cookie does not delete the cookie from your browser. You will need to do this separately within your browser.

If you would like to make changes to your cookie settings, please go to the 'Options' or 'Preferences' menu of your browser. Alternatively, go to the 'Help' option in your browser for more details.

To learn more about the cookie settings for your browser, please select the links below:

- [Internet Explorer](#)
- [Firefox](#)
- [Chrome](#)
- [Android](#)
- [Safari](#)
- [iOS](#)

If you have disabled one or more analytical cookies, we may still use information collected from cookies prior to your disabled preference being set, however, we will stop using the disabled cookie to collect any further information.

5.22 Contact

- **Developer Support:** developer@lge.com