

Chromecast + HTML5

Thursday, October 18, 2018

1:04 PM

Use Your Android Device as a Wiimote-Style Controller to Play Tennis on Your Chromecast
<https://cord-cutters.gadgethacks.com/how-to/use-your-android-device-as-wiimote-style-controller-play-tennis-your-chromecast-0160917/>

Building a Google Chromecast Game | HTML5 Game Development
<https://html5gamedevelopment.com/2016-02-building-a-google-chromecast-game/>

13 Best Chromecast Games to Play With a Phone or Tablet
<https://www.makeuseof.com/tag/best-chromecast-games/>

A Developer's Guide to Implementing Chromecast - Possible Mobile
<https://possiblemobile.com/2017/04/chromecast-overview/>

Can I create a multiplayer game with a Google Cast Remote Display App?
<https://stackoverflow.com/questions/38363235/can-i-create-a-multiplayer-game-with-a-google-cast-remote-display-app>

Get Started | Cast | Google Developers
<https://developers.google.com/cast/docs/developers>

<https://play.google.com/store/apps/details?id=com.ubisoft.dance.JustDance&hl=en>

Enjoy Just Dance's greatest choreographies without a video game console! You simply need your smartphone as a controller and an internet-connected screen (computer, iPad, Apple TV, Chromecast Smart TV).

App Flow for Android:

Integrate CAF Sender into your Android app

ur
necast or

n

Integrate CAF Sender into your Android app

This developer guide describes how to add Google Cast support to your Android sender using CAF Sender.

Note: We recommend that you first try the following code:

- [Cast Android codelab tutorial](#) - Learn step-by-step how to enable an existing Android video app to use a Google Cast device to cast videos to a TV.
- [CastVideos-android sample app](#) (GitHub) - Run, navigate and view this reference sample app which complies with the [UX Guidelines](#) and [Design Checklist](#).

The mobile device or laptop is the *sender* which controls the playback, and the Google device is the *receiver* which displays the content on the TV.

The *sender framework* refers to the Cast class library binary and associated resources running on the sender. The *sender app* or *Cast app* refers to an app also running on the sender. The *receiver app* refers to the HTML application running on the receiver.

The sender framework uses an asynchronous callback design to inform the sender app of events and to transition between various states of the Cast app life cycle.

App flow

The following steps describe the typical high-level execution flow for a sender Android app.

- The Cast framework automatically starts [MediaRouter](#) device discovery based on the Android lifecycle.
- When the user clicks on the Cast button, the framework presents the Cast dialog with the discovered Cast devices.
- When the user selects a Cast device as a route, the framework attempts to launch the receiver app on the Cast device.
- The framework invokes callbacks in the sender app to confirm that the receiver app was launched.

U

app

roid

ample

Cast

present at
e sender.

f events

app:

ctivity

e list of

eiver app

- The framework creates a communication channel between the sender and receiver apps.
- The framework uses the communication channel to load and control media playback on receiver.
- The framework synchronizes the media playback state between sender and receiver: when user makes sender UI actions, the framework passes those media control requests to the receiver and when the receiver sends media status updates, the framework updates the state of the sender UI.
- When the user clicks on the Cast button to disconnect from the Cast device, the framework disconnects the sender app from the receiver.

Framework for HTML5

Integrate CAF Sender into your Chrome app

This developer guide describes how to add Google Cast support to your Chrome sender app using CAF Sender.

Note: We recommend that you first try the following code:

- [CastVideos sample app](#) - provides an implementation of an HTML5/Javascript media player that operates as a Chrome sender. It is fully compliant with our UX guidelines, and demonstrates typical features such as transitioning between local and remote playback while synchronizing progress bar between the two, as well as auto-join during page reload.

Terminology

the

en the
receiver,
e sender

ork will

p

r app

player that
trates
nizing the

The mobile device or laptop is the *sender*, which controls the playback; the Google Cast device is the *receiver*, which displays the content on the screen for playback.

The Chrome Sender API consists of two parts: the Framework API ([cast.framework](#)) and the Base API ([chrome.cast](#)). In general, you make calls on the simpler, higher-level Framework API, which are then processed by the lower-level Base API.

The *sender framework* refers to the Framework API, module and associated resources that provide a wrapper around lower-level functionality. The *sender application* or *Google Cast Chrome application* refers to a web (HTML/JavaScript) app running inside a Chrome browser on a sender device. A *Chrome Receiver application* refers to an HTML/JavaScript application running on Chromecast or a Google Cast device.

The sender framework uses an asynchronous callback design to inform the sender app of events and to transition between various states of the Cast app life cycle.

Load the library

For your app to implement the features of Google Cast, it needs to know the location of the Google Cast Chrome Sender API library, as shown. Add the *loadCastFramework* URL parameter to load the Cast Sender Framework API as well. All pages of your app must load the library as follows:

```
<script src="https://www.gstatic.com/cv/js/sender/v1/cast_sender.js?loadCastFramework=1"></script>
```

Non-JavaScript player

Your sender application's properties and functions may be accessed or used with a non-JavaScript player such as Flash or Silverlight. These frameworks provide methods that wrap your JavaScript sender code and let you pass parameters and handle events in the Flash or Silverlight environment.

ast device

nd the
work API,

s that
gle Cast
e browser

of events

f the
L query
refer to

pt>

hook into
or

Silverlight player.

For example, you can use the ActionScript method `ExternalInterface.call(external-JavaScript-function, parameter-1, parameter-2)` to hook into your media control methods, such as `initializeCastApi`. Likewise, use the ActionScript method `ExternalInterface.addCallback("ExternalInterfaceFunctionName", new ActionScriptAction)` to handle events like button clicks and progress bar updates.

See your framework's documentation for more information about hooking into your server's JavaScript code.

Framework

The Cast Sender Framework API uses the `cast.framework.*` namespace. The namespace represents the following:

- Methods or functions that invoke operations on the API
- Event listeners for listener functions in the API

The framework consists of these main components:

1. The [CastContext](#) is a singleton object that provides information about the current Cast session and triggers events for Cast state and Cast session state changes.
2. The [CastSession](#) object manages the session -- it provides state information and triggers events for state changes such as changes to device volume, mute state, and application metadata.
3. The Cast button element, which is a simple HTML custom element that extends the HTML button. If the provided Cast button is not sufficient, you can use the Cast state to implement a custom Cast button.
4. The RemotePlayerController provides the data binding to simplify implementation of the remote player.

ot-

VENT",

nder's

pace

state, and

s events,

TML
ment a

he remote