MediaRouter API

In this document

- The media route button
  - Use an AppCompatActivity
  - Define the media route button menu item
  - Create a MediaRouteSelector
  - Add the media route button to the action bar
- MediaRouter callbacks
- Controlling a remote playback route

Key classes

- `MediaRouter`
- `MediaRouter.Callback`
- `MediaRouteProvider`

Related samples

- MediaRouter

In order to use the MediaRouter framework within your app, you must get an instance of the `MediaRouter` object and attach a `MediaRouter.Callback` object to listen for routing events. Content sent over a media route passes through the route's associated `MediaRouteProvider` (except in a few special cases, such as a Bluetooth output device). Figure 1 provides a high-level view of the classes used to route content between devices.

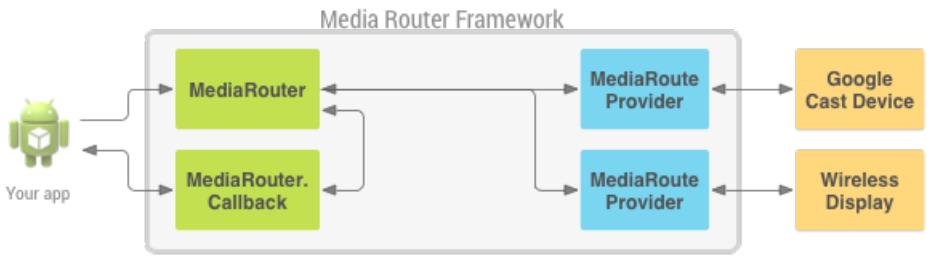

Figure 1. Overview of key media router classes used by apps.

> **Note:** If you want your app to support Google Cast devices, you should use the Cast SDK and build your app as a Cast sender. Follow the directions in the Cast documentation instead of using the MediaRouter framework directly.

The media route button

Android apps should use a media route button to control media routing. The MediaRouter framework provides a standard interface for the button, which helps users recognize and use routing when it's available. The media route button is usually placed on the right side of your app's action bar, as shown in Figure 2.

Figure 2. Media route button in the action bar.

When the user presses the media route button, the available media routes appear in a list as shown in figure 3.

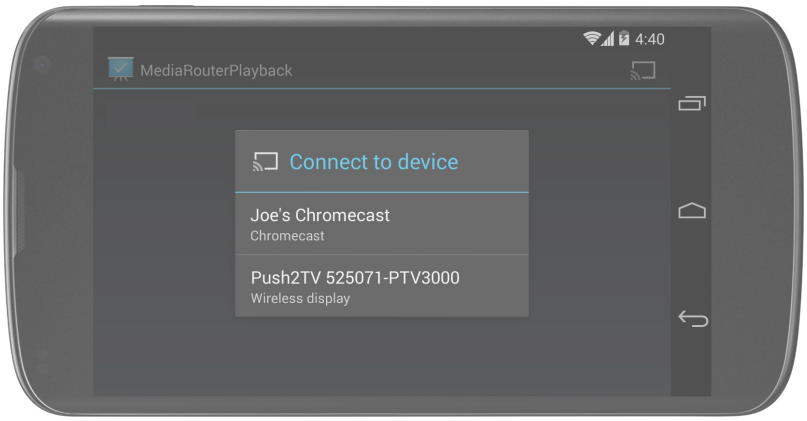

Figure 3. A list of available media routes, shown after pressing the media route button.

Follow these steps to create a media route button:

1. Use an AppCompatActivity
2. Define the media route button menu item
3. Create a MediaRouteSelector
4. Add the media route button to the action bar
5. Create and manage the MediaRouter.Callback methods in your activity's lifecycle

This section describes the first four steps. The next section describes Callback methods.

Use an AppCompatActivity

When you use the media router framework in an activity you should extend the activity from `AppCompatActivity` and import the package `android.support.v7.media`. You must add the v7-appcompat and v7-mediarouter support libraries to your app development project. For more information on adding support libraries to your project, see Support Library Setup.

> **Caution:** Be sure to use the `android.support.v7.media` implementation of the media router framework. Do not use the older `android.media` package.

Define the media route button menu item

Create an xml file that defines a menu item for the media route button. The item's action should be the `MediaRouteActionProvider` class. Here is an example file:

```
// myMediaRouteButtonMenuItem.xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      >

    <item android:id="@+id/media_route_menu_item"
        android:title="@string/media_route_menu_title"
        app:actionProviderClass="android.support.v7.app.MediaRouteActionProvider"
        app:showAsAction="always"
    />
</menu>
```

Create a MediaRouteSelector

The routes that appear in the media route button menu are determined by a `MediaRouteSelector`. Extend your activity from `AppCompatActivity` and build the selector when the activity is created calling `MediaRouteSelector.Builder` from the onCreate() method as shown in the following code sample. Note that the selector is saved in a class variable, and the allowable route types are specified by adding `MediaControlIntent` objects:

```
public class MediaRouterPlaybackActivity extends AppCompatActivity {
    private MediaRouteSelector mSelector;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Create a route selector for the type of routes your app supports.
        mSelector = new MediaRouteSelector.Builder()
                // These are the framework-supported intents
                .addControlCategory(MediaControlIntent.CATEGORY_REMOTE_PLAYBACK)
                .build();
    }
}
```

For most applications, the only route type needed is `CATEGORY_REMOTE_PLAYBACK`. This route type treats the device running your app as a remote control. The connected receiver device handles all content data retrieval, decoding, and playback. This is how apps that support Google Cast, like Chromecast, work.

A few manufacturers support a special routing option called "secondary output." With this routing, your media app retrieves, renders, and streams video or music directly to the screen and/or speakers on the selected remote receiver device. Use secondary output to send content to wireless-enabled music systems or video displays. To enable the discovery and selection of these devices, you need to add the `CATEGORY_LIVE_AUDIO` or `CATEGORY_LIVE_VIDEO` control categories to the MediaRouteSelector. You also need to create and handle your own `Presentation` dialog.

Add the media route button to the action bar

With the media route menu and MediaRouteSelector defined, you can now add the media route button to an activity. Override the `onCreateOptionsMenu()` method for each of your activities to add an options menu.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);

    // Inflate the menu and configure the media router action provider.
    getMenuInflater().inflate(R.menu.sample_media_router_menu, menu);

    // Attach the MediaRouteSelector to the menu item
    MenuItem mediaRouteMenuItem = menu.findItem(R.id.media_route_menu_item);
    MediaRouteActionProvider mediaRouteActionProvider =
            (MediaRouteActionProvider)MenuItemCompat.getActionProvider(
            mediaRouteMenuItem);
    // Attach the MediaRouteSelector that you built in onCreate()
    mediaRouteActionProvider.setRouteSelector(mSelector);

    // Return true to show the menu.
    return true;
}
```

For more information about implementing the action bar in your app, see the Action Bar developer guide.

You can also add a media route button as a `MediaRouteButton` in any view. You must attach a MediaRouteSelector to the button using the `setRouteSelector()` method. See the Google Cast Design Checklist for guidelines on incorporating the media route button into your application.

MediaRouter callbacks

All the apps running on the same device share a single `MediaRouter` instance and its routes (filtered per app by the app's MediaRouteSelector). Each activity communicates with the MediaRouter using its own implementation of `MediaRouter.Callback` methods. The MediaRouter calls the callback methods whenever the user selects, changes, or disconnects a route.

There are several methods in the callback that you can override to receive information about routing events. At a minimum, your implementation of the `MediaRouter.Callback` class should override `onRouteSelected()` and `onRouteUnselected()`.

Since the MediaRouter is a shared resource, your app needs to manage its MediaRouter callbacks in response to the usual activity lifecycle callbacks:

- When the activity is created (`onCreate(Bundle)`) grab a pointer to the `MediaRouter` and hold onto it for the lifetime of the app.
- Attach callbacks to MediaRouter when the activity becomes visible (`onStart()`), and detach them when it is hidden (`onStop()`).

The following code sample demonstrates how to create and save the callback object, how to obtain an instance of `MediaRouter`, and how to manage callbacks. Note the use of the `CALLBACK_FLAG_REQUEST_DISCOVERY` flag when attaching the callbacks in `onStart()`. This allows your MediaRouteSelector to refresh the media route button's list of available routes.

```
public class MediaRouterPlaybackActivity extends AppCompatActivity {
    private MediaRouter mMediaRouter;
    private MediaRouteSelector mSelector;

    // Variables to hold the currently selected route and its playback client
    private MediaRoute mRoute;
    private RemotePlaybackClient mRemotePlaybackClient;

    // Define the Callback object and its methods, save the object in a class variable
    private final MediaRouter.Callback mMediaRouterCallback =
            new MediaRouter.Callback() {

        @Override
        public void onRouteSelected(MediaRouter router, RouteInfo route) {
            Log.d(TAG, "onRouteSelected: route=" + route);

            if (route.supportsControlCategory(
                MediaControlIntent.CATEGORY_REMOTE_PLAYBACK)){
                // Stop local playback (if necessary)
                // ...
```

```
                // Save the new route
                mRoute = route;

                // Attach a new playback client
                mRemotePlaybackClient = new RemotePlaybackClient(this, mRoute);

                // Start remote playback (if necessary)
                // ...
            }
        }

        @Override
        public void onRouteUnselected(MediaRouter router, RouteInfo route, int reason) {
            Log.d(TAG, "onRouteUnselected: route=" + route);

            if (route.supportsControlCategory(
                MediaControlIntent.CATEGORY_REMOTE_PLAYBACK)){

                // Changed route: tear down previous client
                if (mRoute != null && mRemotePlaybackClient != null) {
                    mRemotePlaybackClient.release();
                    mRemotePlaybackClient = null;
                }

                // Save the new route
                mRoute = route;

                if (reason != MediaRouter.UNSELECT_REASON_ROUTE_CHANGED) {
                    // Resume local playback  (if necessary)
                    // ...
                }
            }
        }
    }


    // Retain a pointer to the MediaRouter
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Get the media router service.
        mMediaRouter = MediaRouter.getInstance(this);
        ...
    }

    // Use this callback to run your MediaRouteSelector to generate the list of available media routes
    @Override
    public void onStart() {
        mMediaRouter.addCallback(mSelector, mMediaRouterCallback,
                MediaRouter.CALLBACK_FLAG_REQUEST_DISCOVERY);
        super.onStart();
    }

    // Remove the selector on stop to tell the media router that it no longer
    // needs to discover routes for your app.
    @Override
    public void onStop() {
        mMediaRouter.removeCallback(mMediaRouterCallback);
        super.onStop();
    }
    ...
}
```

The media router framework also provides a `MediaRouteDiscoveryFragment` class, which takes care of adding and removing the callback for an activity.

> **Note:** If you are writing a music playback app and want the app to play music while it is in the background, you must build a `Service` for playback and call the media router framework from the Service's lifecycle callbacks.

Controlling a remote playback route

When you select a remote playback route your app acts as a remote control. The device at the other end of the route handles all content data retrieval, decoding, and playback functions. The controls in your app's UI communicate with the receiver device using a `RemotePlaybackClient` object.

The `RemotePlaybackClient` class provides additional methods for managing content playback. Here are a few of the key playback methods from the `RemotePlaybackClient` class:

- `play()` — Play a specific media file, specified by a `Uri`.
- `pause()` — Pause the currently playing media track.
- `resume()` — Continue playing the current track after a pause command.
- `seek()` — Move to a specific position in the current track.
- `release()` — Tear down the connection from your app to the remote playback device.

You can use these methods to attach actions to the playback controls that you provide in your app. Most of these methods also allow you to include a callback object so you can monitor the progress of the playback task or control request.

The `RemotePlaybackClient` class also supports queueing of multiple media items for playback and management of the media queue.