Media Route Provider API

In this document

Key classes

- `MediaRouteProvider`
- `MediaRouteProviderDescriptor`
- `RouteController`

Related samples

- MediaRouter

The Android media router framework allows manufacturers to enable playback on their devices through a standardized interface called a `MediaRouteProvider`. A route provider defines a common interface for playing media on a receiver device, making it possible to play media on your equipment from any Android application that supports media routes.

This guide discusses how to create a media route provider for a receiver device and make it available to other media playback applications that run on Android.

Overview

The Android media router framework enables media app developers and media playback device manufacturers to connect through a common API and common user interface. App developers that implement a `MediaRouter` interface can then connect to the framework and play content to devices that participate in the media router framework. Media playback device manufacturers can participate in the framework by publishing a `MediaRouteProvider` that allows other applications to connect to and play media on the receiver devices. Figure 1 illustrates how an app connects to a receiving device through the media router framework.

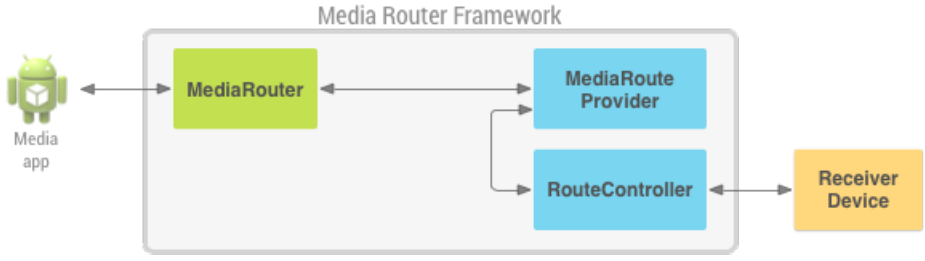

Figure 1. Overview of how media route provider classes provide communication from a media app to a receiver device.

When you build a media route provider for your receiver device, the provider serves the following purposes:

- Describe and publish the capabilities of the receiver device so other apps can discover it and use its playback features.
- Wrap the programming interface of the receiver device and its communication transport mechanisms to make the device compatible with the media router framework.

Distribution of route providers

A media route provider is distributed as part of an Android app. Your route provider can be made available to other apps by extending `MediaRouteProviderService` or wrapping your implementation of `MediaRouteProvider` with your own service and declaring an intent filter for the media route provider. These steps allow other apps to discover and make use of your media route.

Note: The app containing the media route provider can also include a MediaRouter interface to the route provider, but this is not required.

Media router library

The media router APIs are defined in the v7-mediarouter support library. You must add this library 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.

Creating a Provider Service

The media router framework must be able to discover and connect to your media route provider to allow other applications to use your route. In order to do this, the media router framework looks for apps that declare a media route provider intent action. When another app wants to connect to your provider, the framework must be able to invoke and connect to it, so your provider must be encapsulated in a `Service`.

The following example code shows the declaration of a media route provider service and the intent filter in a manifest, which allows it to be discovered and used by the media router framework:

```
<service android:name=".provider.SampleMediaRouteProviderService"
    android:label="@string/sample_media_route_provider_service"
    android:process=":mrp">
    <intent-filter>
        <action android:name="android.media.MediaRouteProviderService" />
    </intent-filter>
</service>
```

This manifest example declares a service that wraps the actual media route provider classes. The Android media router framework provides the `MediaRouteProviderService` class for use as a service wrapper for media route providers. The following example code demonstrates how to use this wrapper class:

```
public class SampleMediaRouteProviderService extends MediaRouteProviderService {

    @Override
    public MediaRouteProvider onCreateMediaRouteProvider() {
        return new SampleMediaRouteProvider(this);
    }
}
```

Specifying Route Capabilities

Apps connecting to the media router framework can discover your media route through your app's manifest declarations, but they also need to know the capabilities of the media routes you are providing. Media routes can be of different types and have different features, and other apps need to be able to discover these details to determine if they are compatible with your route.

The media router framework allows you to define and publish the capabilities of your media route through `IntentFilter` objects, `MediaRouteDescriptor` objects and a `MediaRouteProviderDescriptor`. This section explains how to use these classes to publish the details of your media route for other apps.

Route categories

As part of the programmatic description of your media route provider, you must specify whether your provider supports remote playback, secondary output, or both. These are the route categories provided by the media router framework:

- `CATEGORY_LIVE_AUDIO` — Output of audio to a secondary output device, such as a wireless-enabled music system.
- `CATEGORY_LIVE_VIDEO` — Output of video to a secondary output device, such as Wireless Display devices.
- `CATEGORY_REMOTE_PLAYBACK` — Play video or audio on a separate device which handles media retrieval, decoding, and playback, such as Chromecast devices.

In order to include these settings in a description of your media route, you insert them into an `IntentFilter` object, which you later add to a `MediaRouteDescriptor` object:

```
public final class SampleMediaRouteProvider extends MediaRouteProvider {
    private static final ArrayList<IntentFilter> CONTROL_FILTERS_BASIC;
    static {
        IntentFilter videoPlayback = new IntentFilter();
        videoPlayback.addCategory(MediaControlIntent.CATEGORY_REMOTE_PLAYBACK);
        CONTROL_FILTERS_BASIC = new ArrayList<IntentFilter>();
        CONTROL_FILTERS_BASIC.add(videoPlayback);
    }
}
```

If you specify the `CATEGORY_REMOTE_PLAYBACK` intent, you must also define what media types and playback controls are supported by your media route provider. The next section describes how to specify these settings for your device.

Media types and protocols

A media route provider for a remote playback device must specify the media types and transfer protocols it supports. You specify these settings using the `IntentFilter` class and the `addDataScheme()` and `addDataType()` methods of that object. The following code snippet demonstrates how to define an intent filter for supporting remote video playback using http, https, and Real Time Streaming Protocol (RTSP):

```
public final class SampleMediaRouteProvider extends MediaRouteProvider {

    private static final ArrayList<IntentFilter> CONTROL_FILTERS_BASIC;

    static {
        IntentFilter videoPlayback = new IntentFilter();
        videoPlayback.addCategory(MediaControlIntent.CATEGORY_REMOTE_PLAYBACK);
        videoPlayback.addAction(MediaControlIntent.ACTION_PLAY);
        videoPlayback.addDataScheme("http");
        videoPlayback.addDataScheme("https");
        videoPlayback.addDataScheme("rtsp");
        addDataTypeUnchecked(videoPlayback, "video/*");
        CONTROL_FILTERS_BASIC = new ArrayList<IntentFilter>();
        CONTROL_FILTERS_BASIC.add(videoPlayback);
    }
    ...

    private static void addDataTypeUnchecked(IntentFilter filter, String type) {
        try {
            filter.addDataType(type);
        } catch (MalformedMimeTypeException ex) {
            throw new RuntimeException(ex);
        }
    }
}
```

Playback controls

A media route provider that offers remote playback must specify the types of media controls it supports. These are the general types of control that media routes can provide:

- **Playback controls**, such as play, pause, rewind, and fast-forward.
- **Queuing features**, which allow the sending app to add and remove items from a playlist which is maintained by the receiver device.
- **Session features**, which prevent sending apps from interfering with each other by having the receiver device provide a session id to the requesting app and then checking that id with each subsequent playback control request.

The following code example demonstrates how to construct an intent filter for supporting basic media route playback controls:

```
public final class SampleMediaRouteProvider extends MediaRouteProvider {
    private static final ArrayList<IntentFilter> CONTROL_FILTERS_BASIC;
    static {
        ...
        IntentFilter playControls = new IntentFilter();
        playControls.addCategory(MediaControlIntent.CATEGORY_REMOTE_PLAYBACK);
        playControls.addAction(MediaControlIntent.ACTION_SEEK);
        playControls.addAction(MediaControlIntent.ACTION_GET_STATUS);
        playControls.addAction(MediaControlIntent.ACTION_PAUSE);
        playControls.addAction(MediaControlIntent.ACTION_RESUME);
        playControls.addAction(MediaControlIntent.ACTION_STOP);
        CONTROL_FILTERS_BASIC = new ArrayList<IntentFilter>();
        CONTROL_FILTERS_BASIC.add(videoPlayback);
        CONTROL_FILTERS_BASIC.add(playControls);
    }
    ...
}
```

For more information about the available playback control intents, see the `MediaControlIntent` class.

MediaRouteProviderDescriptor

After defining the capabilities of your media route using `IntentFilter` objects, you can then create a descriptor object for publishing to the Android media router framework. This descriptor object contains the specifics of your media route's capabilities so that other applications can determine how to interact with your media route.

The following example code demonstrates how to add the previously created intent filters to a `MediaRouteProviderDescriptor` and set the descriptor for use by the media router framework:

```
public SampleMediaRouteProvider(Context context) {
    super(context);
    publishRoutes();
}

private void publishRoutes() {
    Resources r = getContext().getResources();
    // Create a route descriptor using previously created IntentFilters
    MediaRouteDescriptor routeDescriptor = new MediaRouteDescriptor.Builder(
            VARIABLE_VOLUME_BASIC_ROUTE_ID,
            r.getString(R.string.variable_volume_basic_route_name))
            .setDescription(r.getString(R.string.sample_route_description))
            .addControlFilters(CONTROL_FILTERS_BASIC)
            .setPlaybackStream(AudioManager.STREAM_MUSIC)
            .setPlaybackType(MediaRouter.RouteInfo.PLAYBACK_TYPE_REMOTE)
            .setVolumeHandling(MediaRouter.RouteInfo.PLAYBACK_VOLUME_VARIABLE)
            .setVolumeMax(VOLUME_MAX)
            .setVolume(mVolume)
            .build();
    // Add the route descriptor to the provider descriptor
    MediaRouteProviderDescriptor providerDescriptor =
            new MediaRouteProviderDescriptor.Builder()
            .addRoute(routeDescriptor)
            .build();

    // Publish the descriptor to the framework
    setDescriptor(providerDescriptor);
}
```

For more information on the available descriptor settings, see the reference documentation for `MediaRouteDescriptor` and `MediaRouteProviderDescriptor`.

Controlling Routes

When an application connects to your media route provider, the provider receives playback commands through the media router framework sent to your route by other apps. To handle these requests, you must provide an implementation of a `MediaRouteProvider.RouteController` class, which processes the commands and handles the actual communication to your receiver device.

The media router framework calls the `onCreateRouteController()` method of your route provider to obtain an instance of this class and then routes requests to it. These are the key methods of the `MediaRouteProvider.RouteController` class, which you must implement for your media route provider:

- `onSelect()` — Called when an application selects your route for playback. You use this method to do any preparation work that may be required before media playback begins.
- `onControlRequest()` — Sends specific playback commands to the receiving device.
- `onSetVolume()` — Sends a request to the receiving device to set the playback volume to a specific value.
- `onUpdateVolume()` — Sends a request to the receiving device to modify the playback volume by a specified amount.
- `onUnselect()` — Called when an application unselects a route.
- `onRelease()` — Called when the route is no longer needed by the framework, allowing it to free its resources.

All playback control requests, except for volume changes, are directed to the `onControlRequest()` method. Your implementation of this method must parse the control requests and respond to them appropriately. Here is an example implementation of this method which processes commands for a remote playback media route:

```
private final class SampleRouteController extends
        MediaRouteProvider.RouteController {
    ...

    @Override
    public boolean onControlRequest(Intent intent, ControlRequestCallback callback) {

        String action = intent.getAction();

        if (intent.hasCategory(MediaControlIntent.CATEGORY_REMOTE_PLAYBACK)) {
            boolean success = false;
            if (action.equals(MediaControlIntent.ACTION_PLAY)) {
                success = handlePlay(intent, callback);
            } else if (action.equals(MediaControlIntent.ACTION_ENQUEUE)) {
                success = handleEnqueue(intent, callback);
            } else if (action.equals(MediaControlIntent.ACTION_REMOVE)) {
                success = handleRemove(intent, callback);
            } else if (action.equals(MediaControlIntent.ACTION_SEEK)) {
                success = handleSeek(intent, callback);
            } else if (action.equals(MediaControlIntent.ACTION_GET_STATUS)) {
                success = handleGetStatus(intent, callback);
            } else if (action.equals(MediaControlIntent.ACTION_PAUSE)) {
                success = handlePause(intent, callback);
            } else if (action.equals(MediaControlIntent.ACTION_RESUME)) {
                success = handleResume(intent, callback);
            } else if (action.equals(MediaControlIntent.ACTION_STOP)) {
                success = handleStop(intent, callback);
            } else if (action.equals(MediaControlIntent.ACTION_START_SESSION)) {
                success = handleStartSession(intent, callback);
            } else if (action.equals(MediaControlIntent.ACTION_GET_SESSION_STATUS)) {
                success = handleGetSessionStatus(intent, callback);
            } else if (action.equals(MediaControlIntent.ACTION_END_SESSION)) {
                success = handleEndSession(intent, callback);
            }

            Log.d(TAG, mSessionManager.toString());
            return success;
        }
        return false;
    }
    ...
}
```

It is important to understand that the `MediaRouteProvider.RouteController` class is intended to act as a wrapper for the API to your media playback equipment. The implementation of the methods in this class is entirely dependent on the programmatic interface provided by your receiving device.