Using Wi-Fi P2P for Service Discovery

PREVIOUS NEXT

THIS LESSON TEACHES YOU TO

1. Set Up the Manifest
2. Add a Local Service
3. Discover Nearby Services

The first lesson in this class, Using Network Service Discovery, showed you how to discover services that are connected to a local network. However, using Wi-Fi Peer-to-Peer (P2P) Service Discovery allows you to discover the services of nearby devices directly, without being connected to a network. You can also advertise the services running on your device. These capabilities help you communicate between apps, even when no local network or hotspot is available.

While this set of APIs is similar in purpose to the Network Service Discovery APIs outlined in a previous lesson, implementing them in code is very different. This lesson shows you how to discover services available from other devices, using Wi-Fi P2P. The lesson assumes that you're already familiar with the Wi-Fi P2P API.

Set Up the Manifest

In order to use Wi-Fi P2P, add the CHANGE_WIFI_STATE, ACCESS_WIFI_STATE, and INTERNET permissions to your manifest. Even though Wi-Fi P2P doesn't require an Internet connection, it uses standard Java sockets, and using these in Android requires the requested permissions.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.nsdchat"
    ...

    <uses-permission
        android:required="true"
        android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.INTERNET"/>
    ...
```

Add a Local Service

If you're providing a local service, you need to register it for service discovery. Once your local service is registered, the framework automatically responds to service discovery requests from peers.

To create a local service:

1. Create a WifiP2pServiceInfo object.
2. Populate it with information about your service.
3. Call addLocalService() to register the local service for service discovery.

```
    private void startRegistration() {
        //  Create a string map containing information about your service.
        Map record = new HashMap();
        record.put("listenport", String.valueOf(SERVER_PORT));
        record.put("buddyname", "John Doe" + (int) (Math.random() * 1000));
        record.put("available", "visible");

        // Service information.  Pass it an instance name, service type
        // _protocol._transportlayer , and the map containing
        // information other devices will want once they connect to this one.
        WifiP2pDnsSdServiceInfo serviceInfo =
                WifiP2pDnsSdServiceInfo.newInstance("_test", "_presence._tcp", record);

        // Add the local service, sending the service info, network channel,
        // and listener that will be used to indicate success or failure of
        // the request.
        mManager.addLocalService(channel, serviceInfo, new ActionListener() {
            @Override
            public void onSuccess() {
                // Command successful! Code isn't necessarily needed here,
                // Unless you want to update the UI or add logging statements.
            }

            @Override
            public void onFailure(int arg0) {
                // Command failed.  Check for P2P_UNSUPPORTED, ERROR, or BUSY
            }
        });
    }
```

Discover Nearby Services

Android uses callback methods to notify your application of available services, so the first thing to do is set those up. Create a WifiP2pManager.DnsSdTxtRecordListener to listen for incoming records. This record can optionally be broadcast by other devices. When one comes in, copy the device address and any other relevant information you want into a data structure external to the current method, so you can access it later. The following example assumes that the record contains a "buddyname" field, populated with the user's identity.

```
final HashMap<String, String> buddies = new HashMap<String, String>();
...
private void discoverService() {
    DnsSdTxtRecordListener txtListener = new DnsSdTxtRecordListener() {
        @Override
        /* Callback includes:
         * fullDomain: full domain name: e.g "printer._ipp._tcp.local."
         * record: TXT record dta as a map of key/value pairs.
         * device: The device running the advertised service.
         */

        public void onDnsSdTxtRecordAvailable(
                String fullDomain, Map record, WifiP2pDevice device) {
                Log.d(TAG, "DnsSdTxtRecord available -" + record.toString());
                buddies.put(device.deviceAddress, record.get("buddyname"));
            }
        };
    ...
}
```

To get the service information, create a WifiP2pManager.DnsSdServiceResponseListener. This receives the actual description and connection information. The previous code snippet implemented a Map object to pair a device address with the buddy name. The service response listener uses this to link the DNS record with the corresponding service information. Once both listeners are implemented, add them to the WifiP2pManager using the setDnsSdResponseListeners() method.

```
private void discoverService() {
...

    DnsSdServiceResponseListener servListener = new DnsSdServiceResponseListener() {
        @Override
        public void onDnsSdServiceAvailable(String instanceName, String registrationType,
                WifiP2pDevice resourceType) {

                // Update the device name with the human-friendly version from
                // the DnsTxtRecord, assuming one arrived.
                resourceType.deviceName = buddies
                        .containsKey(resourceType.deviceAddress) ? buddies
                        .get(resourceType.deviceAddress) : resourceType.deviceName;

                // Add to the custom adapter defined specifically for showing
                // wifi devices.
                WiFiDirectServicesList fragment = (WiFiDirectServicesList) getFragmentManager()
                        .findFragmentById(R.id.frag_peerlist);
                WiFiDevicesAdapter adapter = ((WiFiDevicesAdapter) fragment
                        .getListAdapter());

                adapter.add(resourceType);
                adapter.notifyDataSetChanged();
                Log.d(TAG, "onBonjourServiceAvailable " + instanceName);
        }
    };

    mManager.setDnsSdResponseListeners(channel, servListener, txtListener);
    ...
}
```

Now create a service request and call addServiceRequest(). This method also takes a listener to report success or failure.

```
        serviceRequest = WifiP2pDnsSdServiceRequest.newInstance();
        mManager.addServiceRequest(channel,
                serviceRequest,
                new ActionListener() {
                    @Override
                    public void onSuccess() {
                        // Success!
                    }

                    @Override
                    public void onFailure(int code) {
                        // Command failed.  Check for P2P_UNSUPPORTED, ERROR, or BUSY
                    }
                });
```

Finally, make the call to discoverServices().

```
        mManager.discoverServices(channel, new ActionListener() {

            @Override
            public void onSuccess() {
                // Success!
            }

            @Override
            public void onFailure(int code) {
                // Command failed.  Check for P2P_UNSUPPORTED, ERROR, or BUSY
                if (code == WifiP2pManager.P2P_UNSUPPORTED) {
                    Log.d(TAG, "P2P isn't supported on this device.");
                else if(...)
                    ...
            }
        });
```

If all goes well, hooray, you're done! If you encounter problems, remember that the asynchronous calls you've made take an WifiP2pManager.ActionListener as an argument, and this provides you with callbacks indicating success or failure. To diagnose problems, put debugging code in onFailure(). The error code provided by the method hints at the problem. Here are the possible error values and what they mean

P2P_UNSUPPORTED
: Wi-Fi P2P isn't supported on the device running the app.

BUSY
: The system is to busy to process the request.

ERROR
: The operation failed due to an internal error.

NEXT CLASS: PERFORMING NETWORK OPERATIONS ›

g+1 7

Except as noted, this content is licensed under Creative Commons Attribution 2.5. For details and restrictions, see the Content License.

About Android | Legal | Support