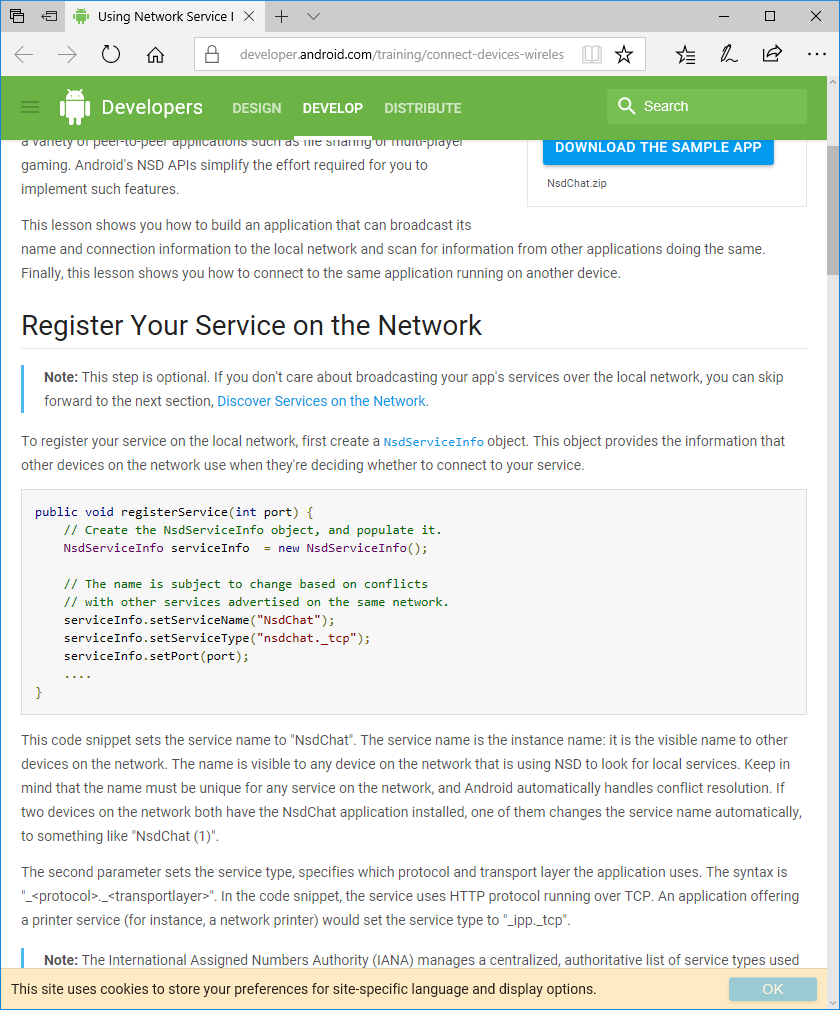
**WiFi**

**Using Network Service Discovery**

**How to register your service on the network**

To register your service on the local network, first create a NsdServiceInfo object. This object provides the information that other devices on the network use when they're deciding whether to connect to your service.

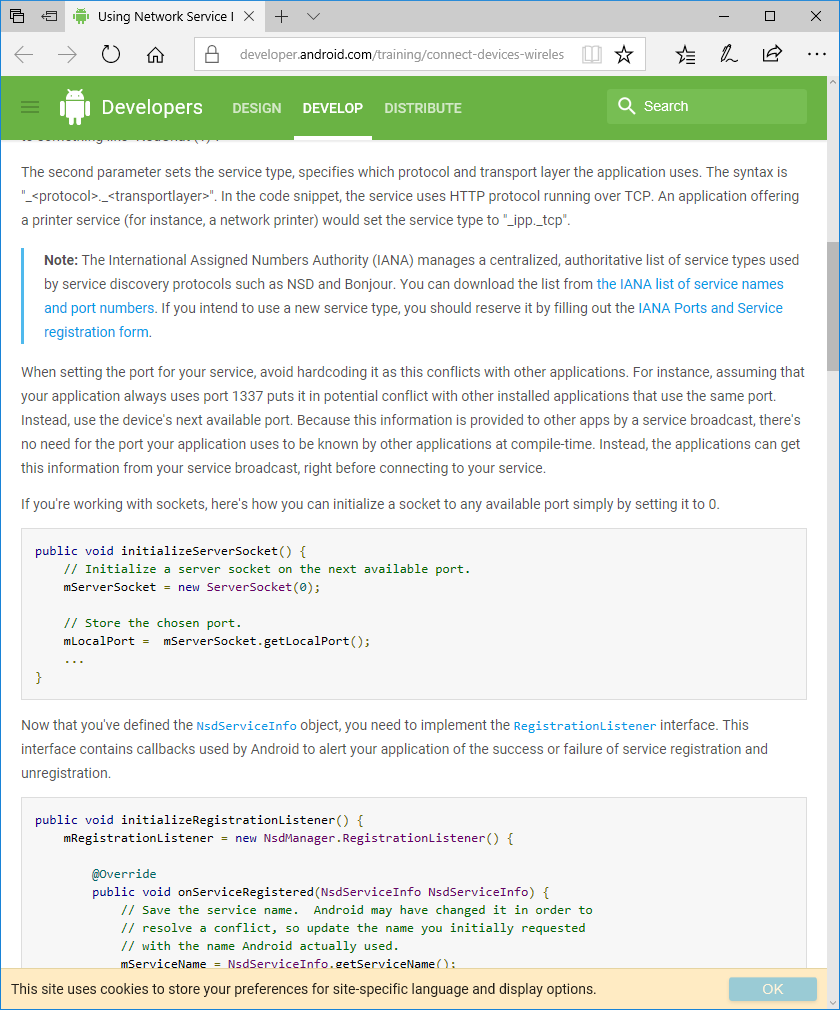


This code snippet sets the service name to "NsdChat". The service name is the instance name: it is the visible name to other devices on the network. The name is visible to any device on the network that is using NSD to look for local services. Keep in mind that the name must be unique for any service on the network, and Android automatically handles conflict resolution. If two devices on the network both have the NsdChat application installed, one of them changes the service name automatically, to something like "NsdChat (1)".

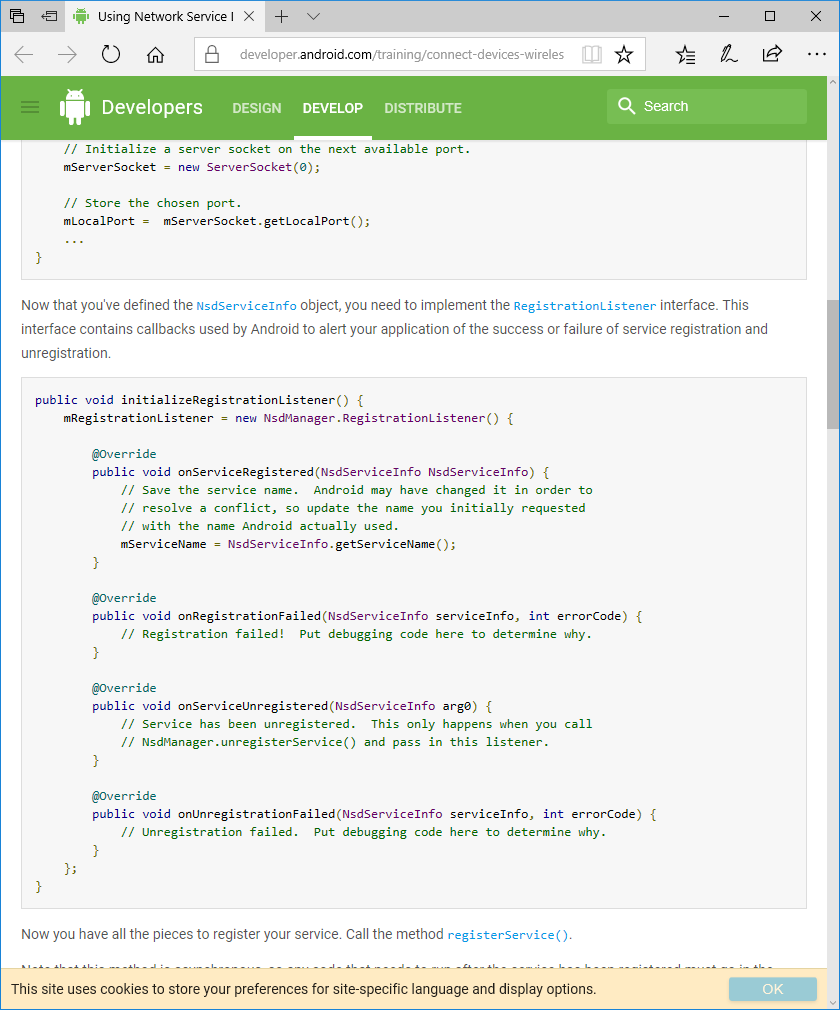
The second parameter sets the service type, specifies which protocol and transport layer the application uses. The syntax is "\_<protocol>.\_<transportlayer>". In the code snippet, the service uses HTTP protocol running over TCP. An application offering a printer service (for instance, a network printer) would set the service type to "\_ipp.\_tcp".

When setting the port for your service, avoid hardcoding it as this conflicts with other applications. For instance, assuming that your application always uses port 1337 puts it in potential conflict with other installed applications that use the same port. Instead, use the device's next available port. Because this information is provided to other apps by a service broadcast, there's no need for the port your application uses to be known by other applications at compile-time. Instead, the applications can get this information from your service broadcast, right before connecting to your service.

If you're working with sockets, here's how you can initialize a socket to any available port simply by setting it to 0.

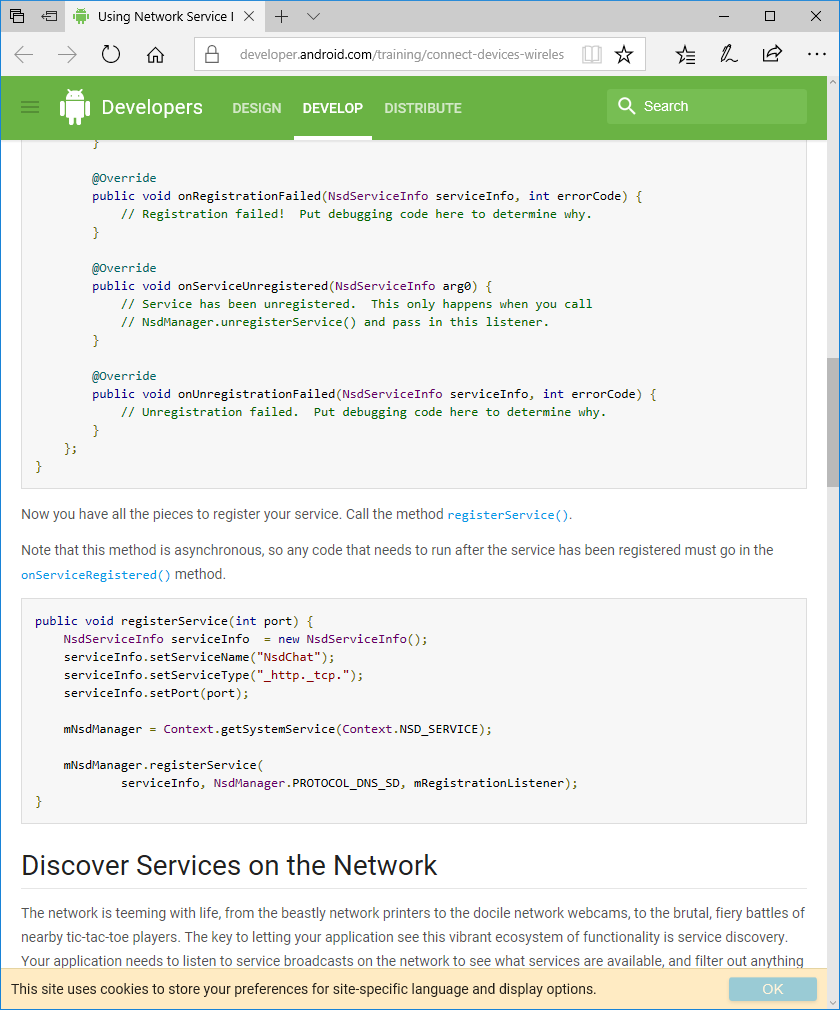


Now that you've defined the NsdServiceInfo object, you need to implement the RegistrationListener interface. This interface contains callbacks used by Android to alert your application of the success or failure of service registration and unregistration.



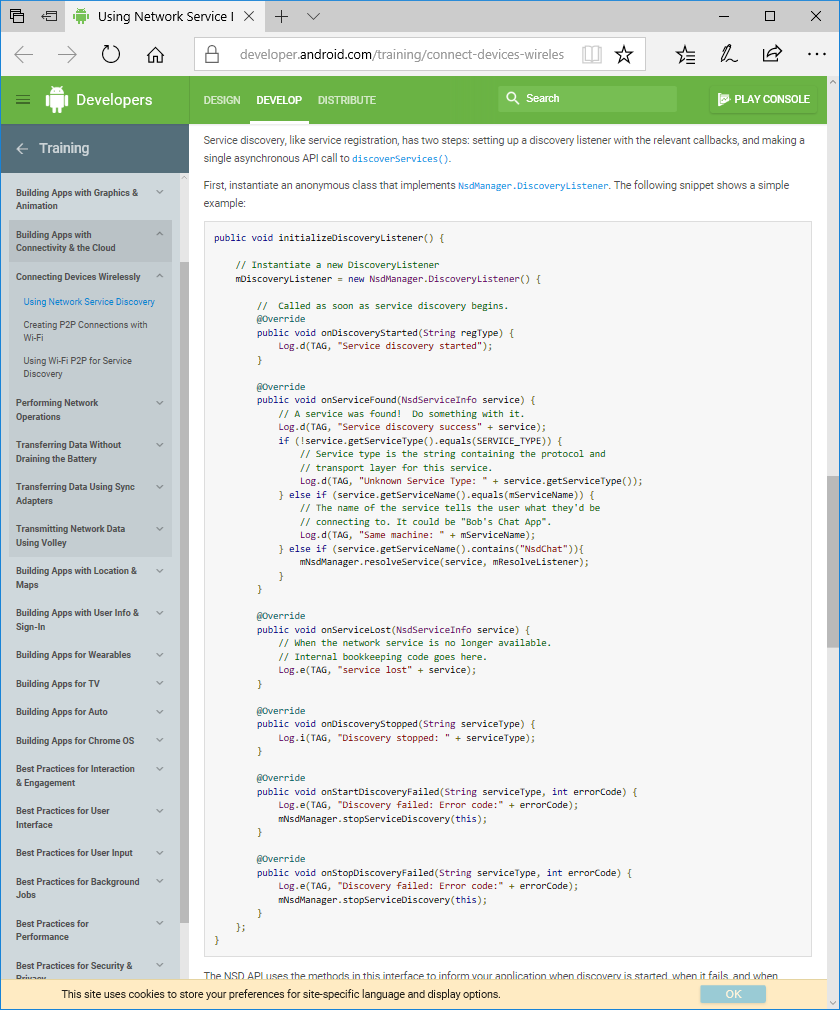
Now you have all the pieces to register your service. Call the method registerService().

Note that this method is asynchronous, so any code that needs to run after the service has been registered must go in the onServiceRegistered() method.



**How to discover services on the network**

Service discovery, like service registration, has two steps: setting up a discovery listener with the relevant callbacks, and making a single asynchronous API call to discoverServices(). First, instantiate an anonymous class that implements NsdManager.DiscoveryListener. The following snippet shows a simple example:

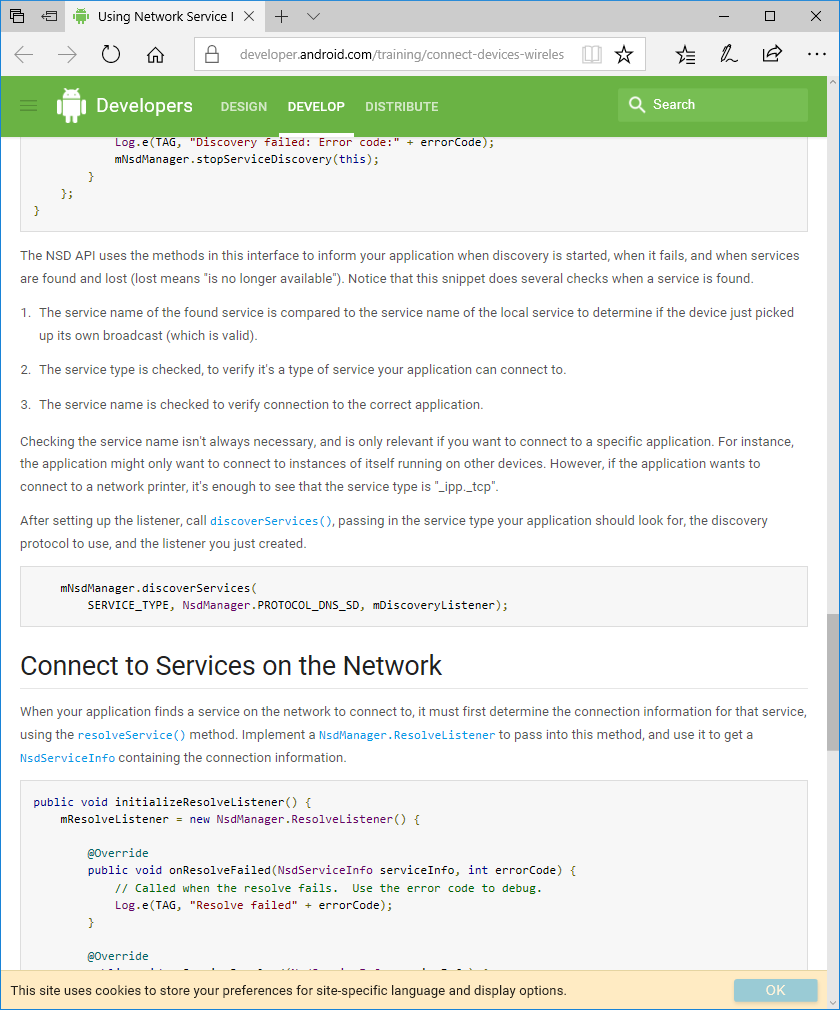


The NSD API uses the methods in this interface to inform your application when discovery is started, when it fails, and when services are found and lost (lost means "is no longer available"). Notice that this snippet does several checks when a service is found.

1. The service name of the found service is compared to the service name of the local service to determine if the device just picked up its own broadcast (which is valid).
2. The service type is checked, to verify it's a type of service your application can connect to.
3. The service name is checked to verify connection to the correct application.

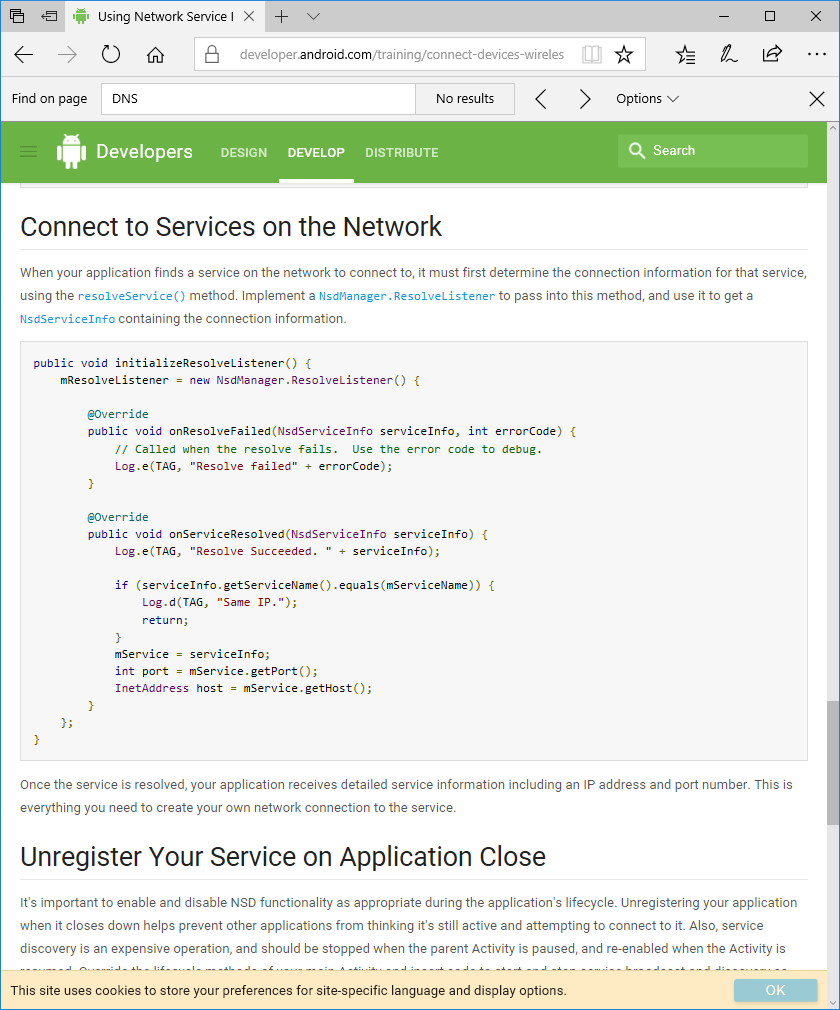
Checking the service name isn't always necessary, and is only relevant if you want to connect to a specific application. For instance, the application might only want to connect to instances of itself running on other devices. However, if the application wants to connect to a network printer, it's enough to see that the service type is "\_ipp.\_tcp".

After setting up the listener, call discoverServices(), passing in the service type your application should look for, the discovery protocol to use, and the listener you just created.



**How to connect to services on the network**

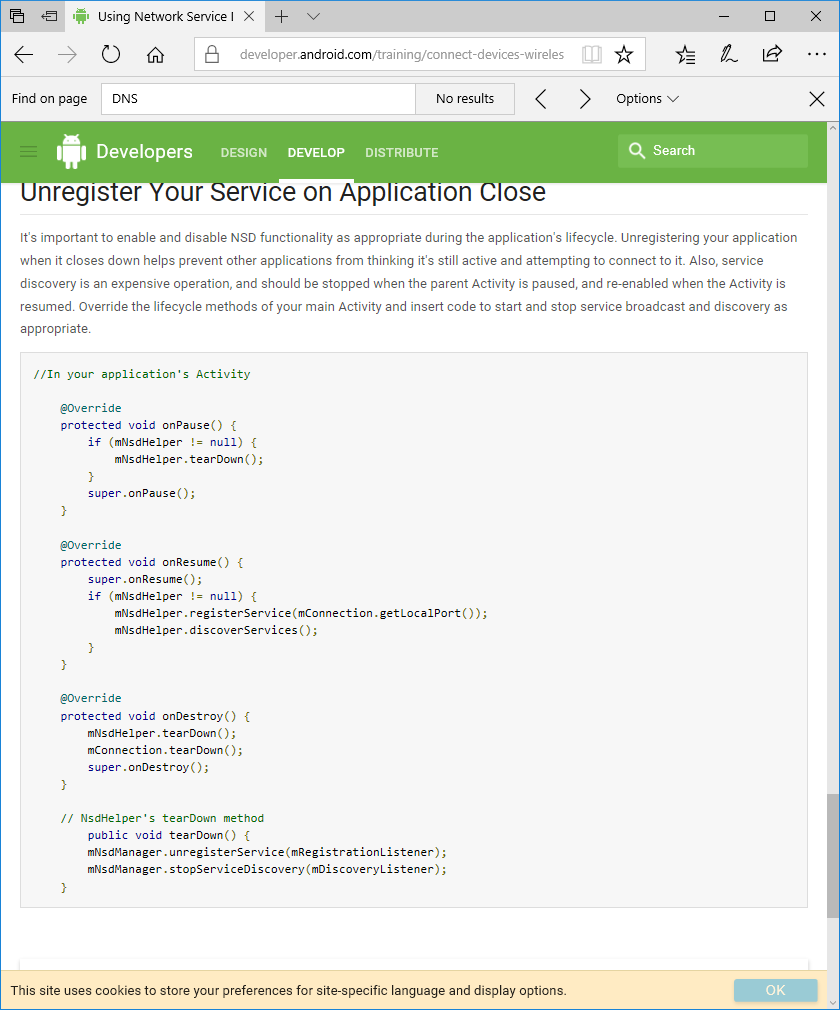
When your application finds a service on the network to connect to, it must first determine the connection information for that service, using the resolveService() method. Implement a NsdManager.ResolveListener to pass into this method, and use it to get a NsdServiceInfo containing the connection information.



Once the service is resolved, your application receives detailed service information including an IP address and port number. This is everything you need to create your own network connection to the service.

**How to unregister a service on application close**

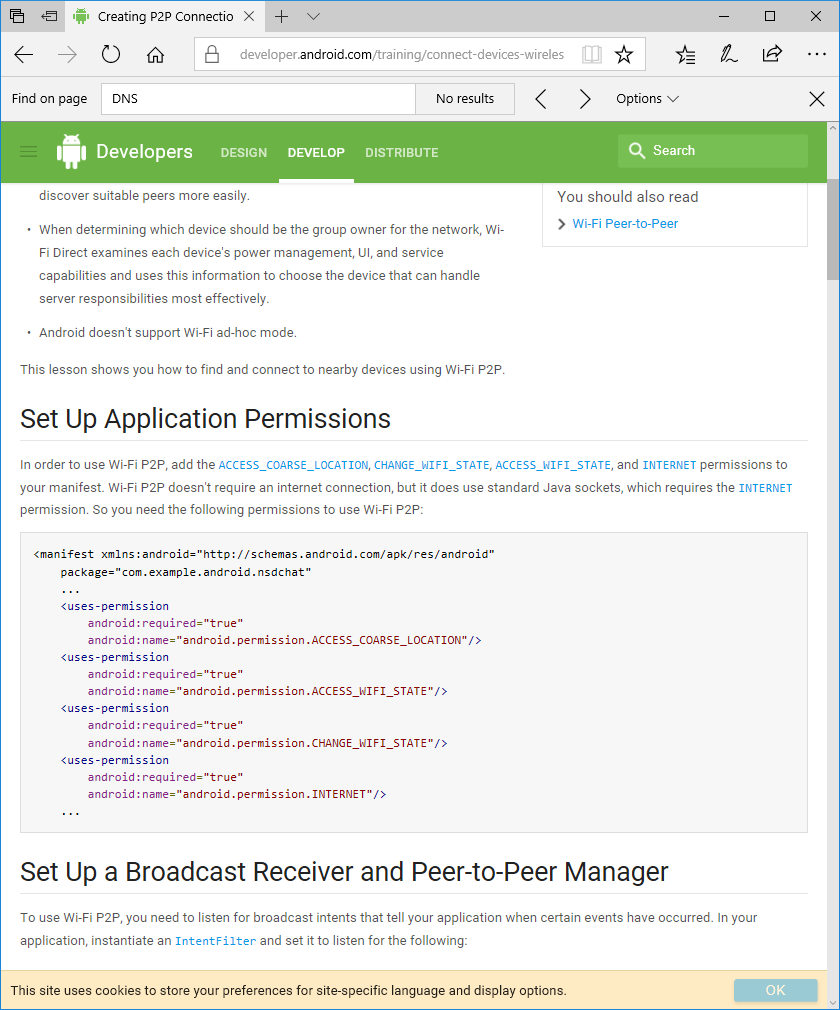
It's important to enable and disable NSD functionality as appropriate during the application's lifecycle. Unregistering your application when it closes down helps prevent other applications from thinking it's still active and attempting to connect to it. Also, service discovery is an expensive operation, and should be stopped when the parent Activity is paused, and re-enabled when the Activity is resumed. Override the lifecycle methods of your main Activity and insert code to start and stop service broadcast and discovery as appropriate.



**Create P2P connection with Wi-Fi**

**How to set up application permissions**

In order to use Wi-Fi P2P, add the ACCESS\_COARSE\_LOCATION, CHANGE\_WIFI\_STATE, ACCESS\_WIFI\_STATE, and INTERNET permissions to your manifest. Wi-Fi P2P doesn't require an internet connection, but it does use standard Java sockets, which requires the INTERNET permission. So you need the following permissions to use Wi-Fi P2P:



**How to set up a broadcast receiver and peer to peer manager**

To use Wi-Fi P2P, you need to listen for broadcast intents that tell your application when certain events have occurred. In your application, instantiate an IntentFilter and set it to listen for the following:

WIFI\_P2P\_STATE\_CHANGED\_ACTION

Indicates whether Wi-Fi P2P is enabled

WIFI\_P2P\_PEERS\_CHANGED\_ACTION

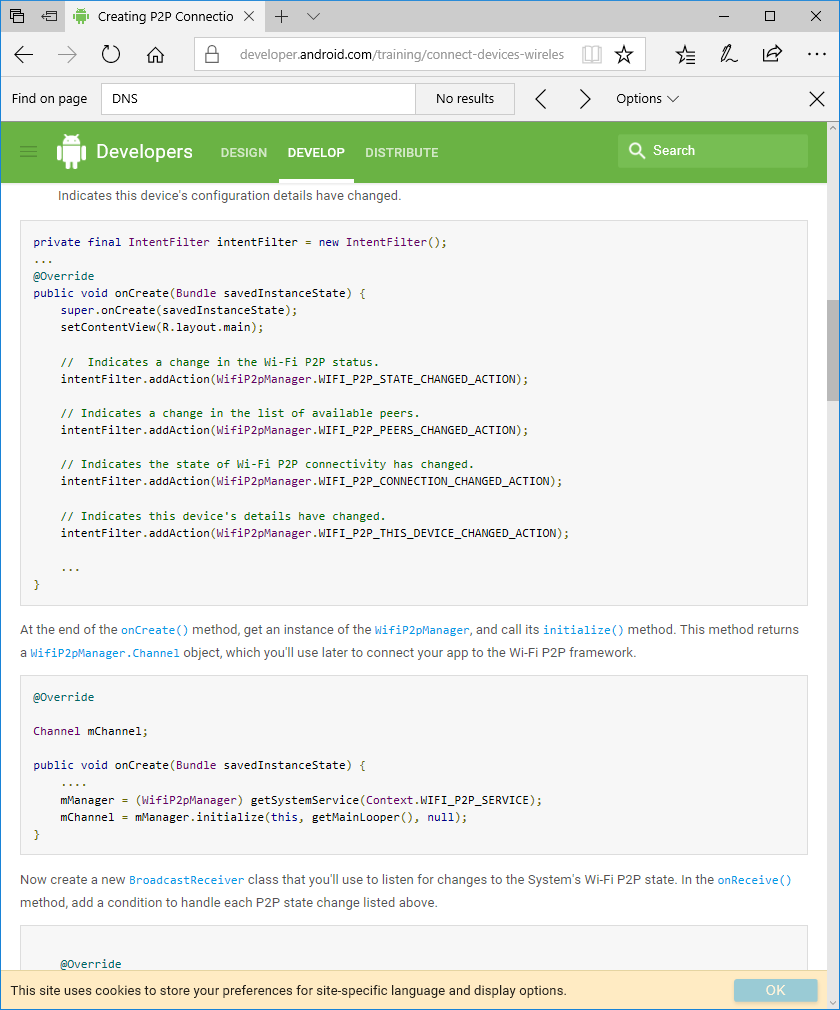
Indicates that the available peer list has changed.

WIFI\_P2P\_CONNECTION\_CHANGED\_ACTION

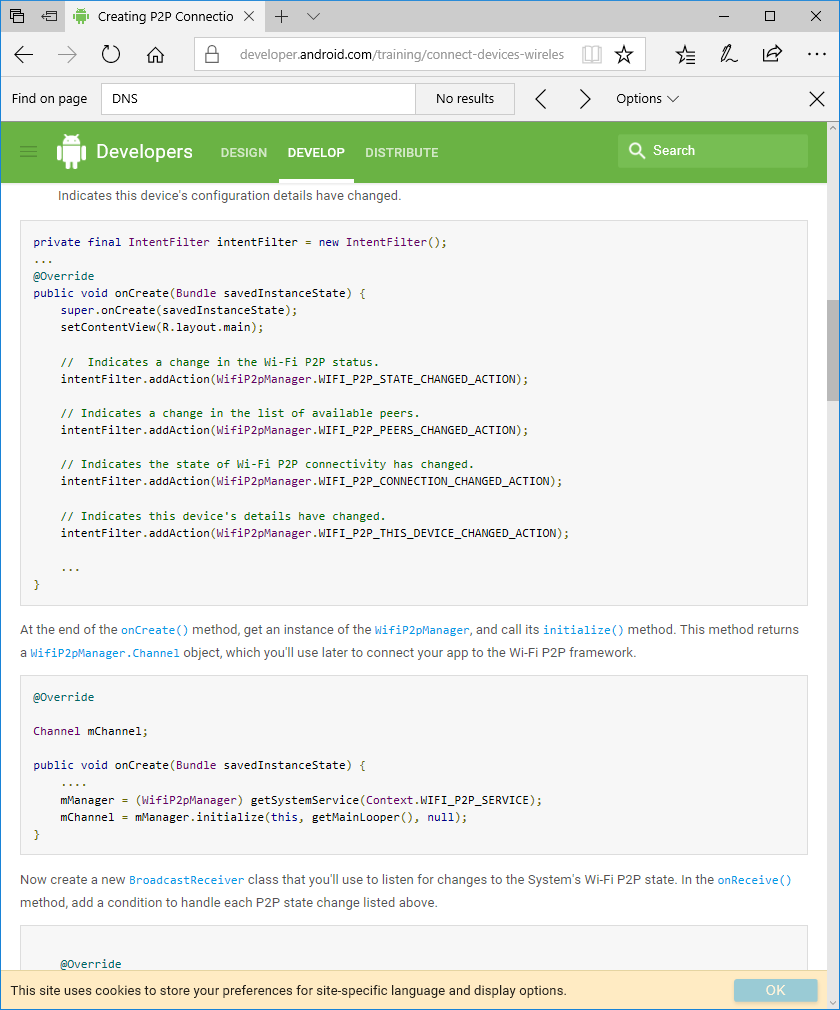
Indicates the state of Wi-Fi P2P connectivity has changed.

WIFI\_P2P\_THIS\_DEVICE\_CHANGED\_ACTION

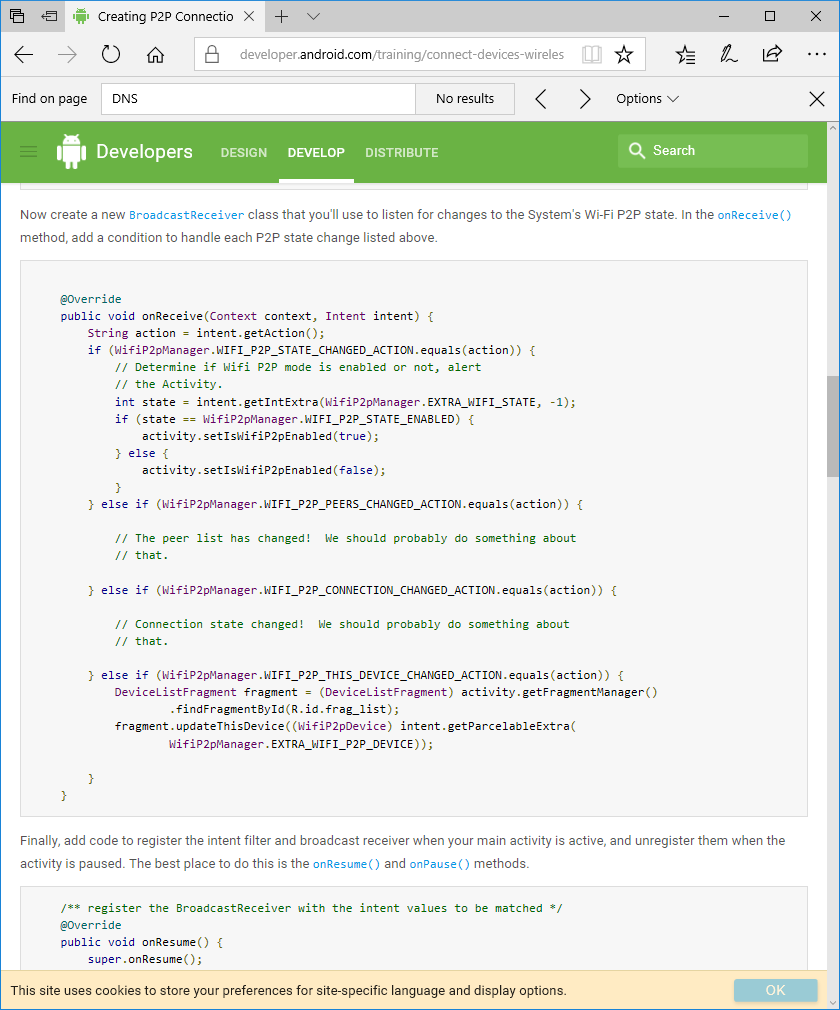
Indicates this device's configuration details have changed.



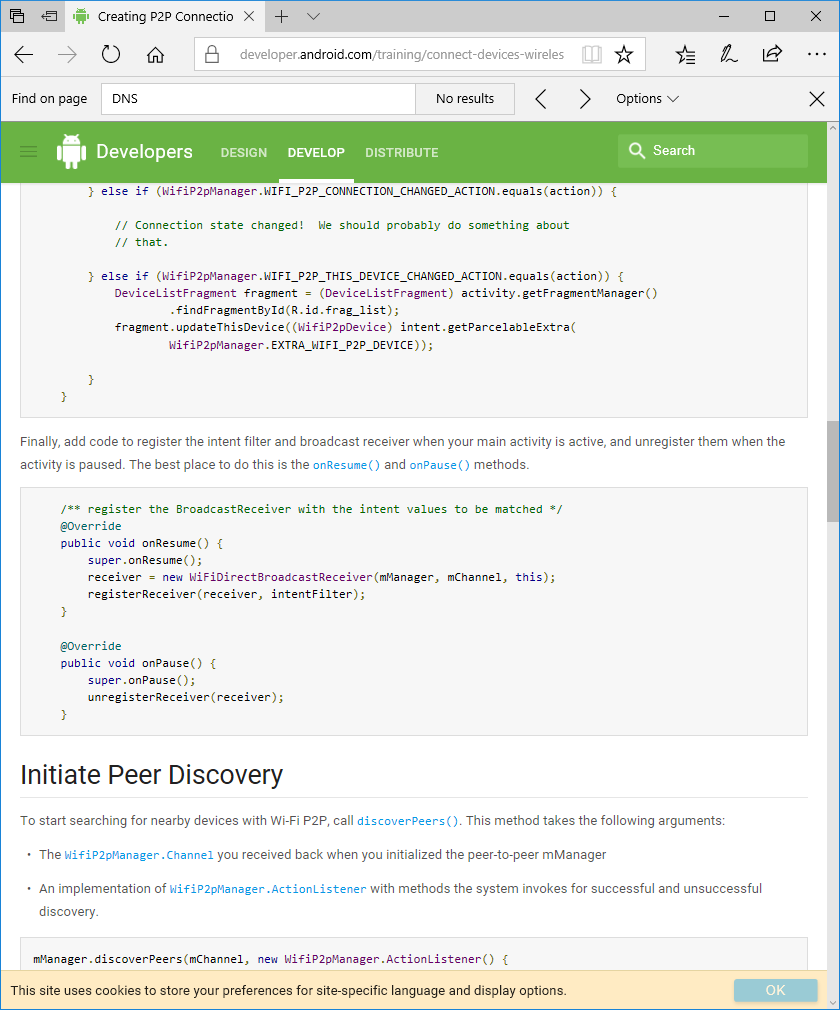
At the end of the onCreate() method, get an instance of the WifiP2pManager, and call its initialize() method. This method returns a WifiP2pManager.Channel object, which you'll use later to connect your app to the Wi-Fi P2P framework.



Now create a new BroadcastReceiver class that you'll use to listen for changes to the System's Wi-Fi P2P state. In the onReceive() method, add a condition to handle each P2P state change listed above.



Finally, add code to register the intent filter and broadcast receiver when your main activity is active, and unregister them when the activity is paused. The best place to do this is the onResume() and onPause() methods.



**How to initiate peer discovery**

**How to fetch list of peers**

**How to connect to a peer**

**Using Wi-Fi P2P for service discovery**

**How to set up the manifest**

**How to add a local service**

**How to discover nearby service**