

APPENDIX A

118116495_1

IXI's Claim Chart regarding Samsung's Infringement of U.S. Patent No. 7,039,033 ¹	
Claim 1	
A system for providing access to the Internet, comprising:	<p>Although the preamble to Claim 1 does not limit the scope of the claim, Samsung provides systems and/or components of systems for providing access to the Internet. Accused systems include Internet Devices,² Samsung Relevant Devices,³ and WLAN Devices.⁴</p> <p>Samsung Relevant Devices provide WLAN Devices with access to the Internet via wireless short range radio signals (e.g., 802.11, Bluetooth) and cellular radio signals. WLAN Devices connect to Samsung Relevant Devices via short distance radio waves using the 802.11 and/or Bluetooth protocols and use the Samsung Relevant Device</p>

¹ IXI provides these infringement contentions for Defendants Samsung Electronics CO., LTD, Samsung Electronics America, Inc., and Samsung Telecommunications America, LLC (collectively, "Samsung"). These contentions contain diagrams, screenshots, and other documentary evidence by way of example and not by way of limitation. These contentions are based on publically available information and in the absence of complete discovery do not represent any claim construction position. IXI reserves the right to amend these contentions as discovery progresses, in response to Samsung's defenses, and in response to any claim construction rulings.

² "Internet Devices" are computers or other devices that function as internet or application servers, including those owned or operated by Samsung or third parties.

³ "Samsung Relevant Devices" include every Samsung smartphone, tablet, laptop or other device that includes a Wireless Hotspot Feature, including but not limited to every Samsung smartphone, Wi-Fi + cellular tablet, and Wi-Fi + cellular laptop that utilizes Windows Phone 7 or above (e.g., ATIV SE, ATIV S Neo, and ATIV Odyssey), every Samsung smartphone, Wi-Fi + cellular tablet, and Wi-Fi + cellular laptop that utilizes Android Version 2.2 Froyo and above (e.g., Charge, Epic 4G, Fascinate, Focus S, Galaxy Ace 3, Galaxy Alpha, Galaxy Appeal, Galaxy Avant, Galaxy Axiom, Galaxy S, Galaxy SII, Galaxy SIII, Galaxy SIII Mini, Galaxy S4, Galaxy S4 Active, Galaxy S4 Mini, Galaxy S5, Galaxy S5 Active, Galaxy S5 Sport, Galaxy S5 Mini, Galaxy S Showcase, Galaxy Gem, Galaxy Legend, Galaxy Exhilarate, Galaxy Exhibit 4G, Galaxy Express, Galaxy Fresh, Galaxy Light, Galaxy Note, Galaxy Note 2, Galaxy Note 3, Galaxy Note 4, Galaxy Note Edge, Galaxy Mega, Galaxy Mega 2, Galaxy Tab (Wi-Fi + Cellular), Galaxy Tab 2 (Wi-Fi + Cellular), Galaxy Tab 3 (Wi-Fi + Cellular), Galaxy Tab 4 (Wi-Fi + Cellular), Galaxy Tab Pro (Wi-Fi + Cellular), Galaxy Tab S (Wi-Fi + Cellular), Galaxy Rugby Pro, Galaxy Metrix 4G, Galaxy Stratosphere II, Gravity SMART T589, Galaxy S Aviator, Galaxy S Blaze 4G, Galaxy S Relay 4G, Galaxy Stellar, Galaxy Victory 4G LTE, Indulge, Infuse 4G, Intercept, Galaxy Nexus, Nexus S, Nexus 10, Rugby Smart, Transform, Vibrant), the LTE Mobile Hotspot PRO and SCH-LC11 4G Mobile Hotspot, and devices reasonably similar in the operation of Wireless Hotspot Features.

"Wireless Hotspot Features" includes the "Portable Wi-Fi Hotspot," "Mobile Hotspot," "Internet Sharing," "Bluetooth Tethering," and/or reasonably similar features that allows a Samsung Relevant Device to wirelessly share (e.g., via Wi-Fi or Bluetooth) its cellular connection with another device.

⁴ "WLAN Devices" include laptops (e.g., Samsung ATIV Book 9 Plus), tablets (e.g., Galaxy Note Tablet, Galaxy Tab, Galaxy Tab 2, Galaxy Tab 3, Galaxy Tab 4, Galaxy Tab, Galaxy Note Tablet), headsets, smart watches (e.g., Galaxy Gear, Gear 2, Gear 2 Neo, Gear Live, Gear S, and Gear Fit), smart TVs, printers, and other devices (including both Samsung and third party devices) that connect to Samsung Relevant Devices via short range radio signals (e.g., Wi-Fi, Bluetooth).


	<p>to access cellular networks and the Internet.</p> <p>For example, all Samsung Relevant Devices include Wireless Hotspot Features. With the exception of the LTE Mobile Hotspot PRO and the SCH-LC11 4G Mobile Hotspot, all Samsung Relevant Devices include the Android Operating System version 2.2 or greater or the Windows Phone Operating System version 7 or greater. These operating system versions each include Wireless Hotspot Features. (See Wired Webpage, “Android 2.2 ‘Froyo’ Features USB, Wi-Fi Tethering”⁵; Windows Phone Webpage, “Share my Connection”⁶)</p> <p>The Samsung Galaxy S4⁵, for example, includes the “Portable Wi-Fi hotspot” feature, which allows WLAN Devices to connect to the Samsung Galaxy S4 via Wi-Fi and use its cellular connection. (See Samsung Galaxy S4 User Manual at 121)</p> <p>Similarly, the Samsung ATIV S Neo⁶ includes the “Internet Sharing” feature, which allows WLAN Devices to connect to the Samsung ATIV Neo via Wi-Fi and use its cellular connection. (See Samsung ATIV S Neo User Manual (Sprint) at 61)</p> <p>The LTE Mobile Hotspot PRO and the SCH-LC11 4G Mobile Hotspot are each themselves mobile hotspots with the primary purpose of providing Wireless Hotspot Features and providing WLAN Devices with access to the Internet.</p>
<p>a first wireless device, in a short distance wireless network, having a software component to access information from the Internet by communicating with a cellular network in response to a first short-range radio signal,</p>	<p>Each Samsung Relevant Device is a first wireless device in a short distance network. The Samsung Relevant Devices each have a software component (e.g., Wireless Hotspot Features software and Android or Windows operating system software) used to access information from the Internet by communicating with a cellular network in response to a first short-range Wi-Fi and/or Bluetooth radio frequency signal received from a WLAN Device.</p> <p>For example, the Wireless Hotspot Features of each Samsung Relevant Device provide WLAN Devices with access to Internet Devices. The Samsung Relevant Devices receive wireless short range radio signals (e.g., 802.11, Bluetooth) from WLAN Devices and in response to these signals retrieve information from the Internet (e.g., website data) via cellular radio signals (e.g., GSM, CDMA, LTE) for relay to the WLAN Devices.</p>

⁵ The Galaxy S4 contains similar components and features as other Samsung Relevant Devices and is representative of the hardware components of these devices.

⁶ The ATIV S Neo contains the Windows Operating System and is representative of the software features on Samsung Relevant Devices utilizing this operating system.

<p>wherein the first wireless device communicates with the cellular network and receives the first short-range radio signal; and,</p>	<p>In addition, other Network Services provided by WLAN Devices to Samsung Relevant Devices utilize the software components of Samsung Relevant Devices to access information from the Internet Devices, such as account authentication, user preferences, cloud-based data (e.g., synchronization, email, calendar, messages, media, etc.), information requested by the Galaxy Gear Watches, and other information when they are connected to Samsung Relevant Devices via Wireless Hotspot Features, Wi-Fi Direct (P2P), Bluetooth, or reasonably similar protocols.⁷</p>
<p>a second wireless device, in the short distance wireless network, to provide the first short-range radio signal,</p>	<p>Each WLAN Device is the claimed second wireless device in a short distance wireless network (e.g., Wi-Fi 802.11 and/or Bluetooth network) that provides the first short-range radio signal to a Samsung Relevant Device. WLAN Devices connect to Samsung Relevant Devices via short distance radio waves using the 802.11 and/or Bluetooth protocol and use the Samsung Relevant Device to access cellular networks and the Internet when using Wireless Hotspot Features or providing other Network Services via Wi-Fi Direct (P2P), Bluetooth, or reasonably similar protocols.</p>
<p>wherein the software component includes a network address translator software component to translate between a first Internet Protocol (“IP”) address provided to the first wireless device from the cellular network and a second address for the second wireless device provided by the first wireless device,</p>	<p>Samsung Relevant Devices include network address translator software components for translating between a first IP addresses provided from the Internet Device over the cellular network and a second IP address for the WLAN Device provided over a Wi-Fi and/or Bluetooth network. <i>See Samsung’s Answers to IXI’s First Set of Requests for Admission, No. 30</i> (“ Samsung admits that Samsung devices with Wi-Fi or Bluetooth hotspot features support Network Address Translation and are compatible with RFC 1631.”).</p> <p>For example, when the Samsung Galaxy S4 is connected to the Internet, it receives an IP address from the cellular network connected to the Internet. When the Samsung Galaxy S4 is in the Portable Hotspot mode, the Samsung Galaxy S4 creates a Wi-Fi network. A local area Wi-Fi network includes a plurality of private addresses; each is provided to a device connected to the Wi-Fi network.</p> <p>One of the most popular methods to implement Wireless Hotspot Features is via Network Address Translation (or NAT), for example for an IP network. Software components in the Samsung Relevant Devices include a network address translator software component to implement the NAT functionality.</p>

⁷ “Network Services” include services such as Wireless Hotspot Features, security, DHCP server functions, DNS server functions, pairing management, virtual private networks, firewalls, monitoring and statistics, health monitoring, gaming (e.g., Group Play), messaging, printing, media-sharing (e.g., via Nearby Devices, AllShare Play, Link, Google Play, Plex, Google+, Facebook), Accessory services Galaxy Gear services, file sharing (e.g., sharing files via Samba, File Explorer, FTP servers, secure shell servers, Dropbox), IANA services, or other services provided to Samsung Relevant Devices from WLAN Devices or vice versa over short distance radio signals, such as those compliant with the Accessory Framework, Chord Framework, AllShare Framework, Media Control Framework, Universal Plug and Play (UPnP), and/or Digital Living Network Alliance (DLNA).

	<p>NAT, or more specifically Network Address and Port Translation (NAPT), involves translating a private IP address to a public IP address and vice versa. More specifically, NAPT involves translating between a public IP address and a plurality of private IP addresses connected to the wireless local area network.</p>
<p>wherein the software component includes a service repository software component to identify a service provided by the second wireless device.</p>	<p>Samsung Relevant Devices include service repository software components for identifying one or more Network Services provided by the WLAN Device(s). The Wireless Hotspot Features on all Samsung Relevant Devices identify the SSID, IP address, and MAC address of a connected WLAN Device, for example, as shown below.</p>  <p>In addition or alternatively, the operating system software on Samsung Relevant Devices includes a service repository software component for identifying Network Services provided by WLAN Devices. For example, all Samsung Relevant Devices with Android versions 4.1 or later include the Network Service Discovery API, which allows end-users and Network Service applications on Samsung Relevant Devices “to identify other devices on the local network that support the services your app requests.” (See Android Developer Webpage, “Using Network Service Discovery”)ⁱⁱⁱ; (See Android Developer Webpage, “Android 4.1 APIs”)^{iv}</p>

Using Network Service Discovery

Adding Network Service Discovery (NSD) to your app allows your users to identify other devices on the local network that support the services your app requests. This is useful for a variety of peer-to-peer applications such as file sharing or multi-player gaming. Android's NSD APIs simplify the effort required for you to implement such features.

This lesson shows you how to build an application that can broadcast its name and connection information to the local network and scan for information from other applications doing the same. Finally, this lesson shows you how to connect to the same application running on another device.

< PREVIOUS

THIS LESSON

1. Register
2. Discover
3. Connect
4. Unregister

Close

(See Android Developer Webpage, "Using Network Service Discovery")^v

Discover Services on the Network

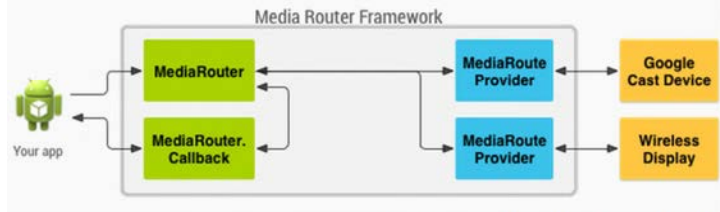
The network is teeming with life, from the beastly network printers to the docile network webcams, to the brutal, fiery battles of nearby tic-tac-toe players. The key to letting your application see this vibrant ecosystem of functionality is service discovery. Your application needs to listen to service broadcasts on the network to see what services are available, and filter out anything the application can't work with.

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:

(See Android Developer Webpage, "Discover Services on the Network")^{vi}

As another example, all Samsung Relevant Devices with Android versions 4.3 or later include the `MediaRouter` and `MediaRouteProvider` framework and APIs, which allows compliant Network Service applications (e.g., media-sharing applications) on Samsung Relevant Devices to identify Network Services, such as displaying and other playback services, on WLAN Devices (e.g., Google Cast Devices and Wireless Displays, below).



(See Android’s Developer Webpage, “Media Route Provider”)^{vii}

MediaRouter allows applications to control the routing of media channels and streams from the current device to external speakers and destination devices. A MediaRouter is retrieved through `Context.getSystemService()` of a `Context.MEDIA_ROUTER_SERVICE`. The media router API is not thread-safe; all interactions with it must be done from the main thread of the process.

(See Android’s Developer Webpage, “MediaRouter”)^{viii}

Users want to play media content from their Android devices bigger, brighter, and louder on connected playback devices such as televisions, stereos, and home theater equipment. As a manufacturer of these devices, allowing Android users to instantly show a picture, play a song, or share a video for friends and family using your product can make it much more compelling and engaging.

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.

(See Android’s Developer Webpage, “Media Route Provider”)^{ix}

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*.

(See Android’s Developer Webpage, “Media Route Provider”)^x

Media route providers are used to publish additional media routes for use within an application. Media route providers may also be declared as a service to publish additional media routes to all applications in the system.

The purpose of a media route provider is to discover media routes that satisfy the criteria specified by the current `MediaRouteDiscoveryRequest` and publish a `MediaRouteProviderDescriptor` with information about each route by calling `onDescriptor(MediaRouteProviderDescriptor)` to notify the currently registered `MediaRouteProvider.Callback`.

The provider should watch for changes to the discovery request by implementing `onDiscoveryRequestChanged(MediaRouteDiscoveryRequest)` and updating the set of routes that it is attempting to discover. It should also handle route control requests such as volume changes or media control intents by implementing `onCreateRouteController(String)` to return a `MediaRouteProvider.RouteController` for a particular route.

A media route provider may be used privately within the scope of a single application process by calling `MediaRouter.addProvider` to add it to the local `MediaRouter`. A media route provider may also be made available globally to all applications by registering a `MediaRouteProviderService` in the provider’s manifest. When the media route provider is registered as a service, all applications that use the media router API will be able to discover and use the provider’s routes without having to install anything else.

(See Android’s Developer Webpage, “Media Route Provider”)^{xi}

As another example, all Samsung Relevant Devices with Android 4.0 or later include the Wi-Fi Peer-to-Peer (P2P) Service Discovery API that allows compliant applications “to discover [i.e., identify] the services of nearby devices directly.” (See Android Developer Webpage, “Using Wi-Fi P2P for Service Discovery”)^{xii}; (See Android Developer Webpage, “Ice Cream Sandwich”)^{xiii}

Using Wi-Fi P2P for Service Discovery

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](#).

< PREVIOUS	NEXT >
THIS LESSON TEACHES YOU TO	
<ol style="list-style-type: none"> 1. Set Up the Manifest 2. Add a Local Service 3. Discover Nearby Services 	

(See Android Developer Webpage, “Using Wi-Fi P2P for Service Discovery”)^{xiv}

All Samsung Relevant Devices with Bluetooth capability include Bluetooth APIs that allows compliant applications to “connect to other devices through service discovery.” (See Android Developer Webpage, “Bluetooth”)^{xv} As shown below, the Android Bluetooth APIs, including at least BluetoothClass, createRfcommSocketToServiceRecord (UUID), and listenUsingRfcommWithServiceRecord(String, UUID), identify services by class and automatically perform service discovery protocol (SDP) requests to identify WLAN Device Network Services. See Samsung’s Answers to IXI’s First Set of Requests for Admission, No. 30 (“Samsung admits that Samsung devices with Bluetooth capability support Bluetooth SDP.”).

Class Overview

Represents a Bluetooth class, which describes general characteristics and capabilities of a device. For example, a Bluetooth class will specify the general device type such as a phone, a computer, or headset, and whether it’s capable of services such as audio or telephony.

Every Bluetooth class is composed of zero or more service classes, and exactly one device class. The device class is further broken down into major and minor device class components.

BluetoothClass is useful as a hint to roughly describe a device (for example to show an icon in the UI), but does not reliably describe which Bluetooth profiles or services are actually supported by a device. Accurate service discovery is done through SDP requests, which are automatically performed when creating an RFCOMM socket with createRfcommSocketToServiceRecord(UUID) and listenUsingRfcommWithServiceRecord(String, UUID)

Use getBluetoothClass() to retrieve the class for a remote device.

(See Android Developer Webpage, “Bluetooth Class”)^{xvi}


All Samsung Relevant Devices with the Windows Phone 8 operating system or later include the Windows.Networking.Proximity namespace, which includes the PeerFinder and PeerWatcher API for identifying Network Services within wireless range.

Proximity for Windows Phone 8

Applies to: Windows Phone 8 and Windows Phone Silverlight 8.1 only

Proximity refers to a set of classes in the Windows Runtime that support connections between devices that are within close range of each other. By using this API, your app can establish a connection through a tap, or by browsing for other devices that are running your app—peer apps—within wireless range. For example, one of these apps might be a multiplayer game in which two users tap their phones together to establish a shared game session. Or, an app might give users the ability to tap a computer and receive a link to a location where they can get more information or make a purchase. Windows Phone 8 supports Proximity communication using Near Field Communication (NFC). This topic provides an overview of the Proximity API for Windows Phone.

(See Microsoft Dev Center Webpage, “Proximity for Windows Phone 8”)^{xvii}

	<h2>PeerFinder class</h2>  <p>Enables you to discover other instances of your app on nearby devices and create a socket connection between the peer apps by using a tap gesture or by browsing. For creating Bluetooth socket connections on Windows 8.1 and later, use <code>Windows.Devices.Bluetooth.Rfcomm</code> instead.</p> <p>(See Microsoft Dev Center Webpage, “PeerFinder class”)^{xviii}</p> <h3>Remarks</h3> <p>You can use the <code>FindAllPeersAsync</code> method to get a list of all peers within range. However, the <code>FindAllPeersAsync</code> method scans for peers once and then completes. Alternatively, you can use the <code>PeerWatcher</code> class to scan for peers and get updates as they are found and incrementally update your list of available peer apps. The <code>PeerWatcher</code> continuously scans for new peer apps within range and removes stale peer apps. You can update your list of peer apps by handling the <code>Added</code> event, which occurs when a new peer app is found, and the <code>Removed</code> event which occurs when a stale peer app is removed. The <code>PeerWatcher</code> continues to scan until you call the <code>Stop</code> method, or the <code>PeerFinder.FindAllPeersAsync</code> or <code>PeerFinder.ConnectAsync</code> methods.</p> <p>(See Microsoft Dev Center Webpage, “PeerWatch class”)^{xix}</p> <p>As another example, Samsung Relevant Devices with Windows Phone 8.1 or later include the Wi-Fi Direct API, which identifies Network Services provide by WLAN Devices.</p> <h3>Remarks</h3> <p>You can use the <code>WiFiDirectDevice</code> class to establish a socket connection with other devices that have a Wi-Fi Direct (WFD) capable device. You can call the <code>GetDeviceSelector</code> method to get the device identifier for a Wi-Fi Direct device. Once you have a reference to a <code>WiFiDirectDevice</code> on your computer, you can call the <code>GetConnectionEndpointPairs</code> method to get an <code>EndpointPair</code> object and establish a socket connection using the <code>Windows.Networking.Sockets</code> API.</p> <p>You can add a handler for the <code>ConnectionStatusChanged</code> event to be notified when the connection has been established or disconnected.</p> <p>Only one app can be connected to a Wi-Fi Direct device at a time.</p> <p>You must enable the <code>Proximity</code> capability to communicate with Wi-Fi Direct devices.</p> <p>(See Microsoft Dev Center, “WiFiDirectDevice class”)^{xx}</p>
--	---

In addition, as shown below, Samsung Relevant Devices with Windows Phone 8 or later include Bluetooth APIs, which identify Network Services provided by WLAN Devices. See Samsung’s Answers to IXI’s First Set of Requests for Admission, No. 30 (“Samsung admits that Samsung devices with Bluetooth capability support Bluetooth SDP.”).

▲ Peer discovery

Discovery is the process of finding a Bluetooth device or app that advertises a service with which you want to interact. In an app-to-device scenario, you can only discover devices that are already paired with the phone on which the app is running. Pairing is the process of using the Bluetooth control panel on your phone to find Bluetooth devices and then connect to them. Pairing typically involves sharing a PIN, or both sides agreeing to connect. In the case of app-to-app Bluetooth connection, one app is looking for another instance of itself on another phone. These phones do not need to be paired for discovery to occur.

(See Microsoft Dev Center, “Bluetooth for Windows Phone 8”)^{xxi}

Further, Samsung Relevant Devices installed with Samsung’s AllShare Framework and/or Media Control Framework also include service repository software components for identifying Network Services provided by WLAN Devices.⁸ As shown below, the Media Control API and AllShare work on higher levels than the Android Framework / API.

⁸ According to Samsung’s webpage, the AllShare Framework was installed on the Samsung Galaxy S4, Samsung Galaxy S3, Samsung Note 10.1, Samsung Note 2, and was be installed by default on later released Samsung smart phones.

The following figure shows the Media Control architecture.

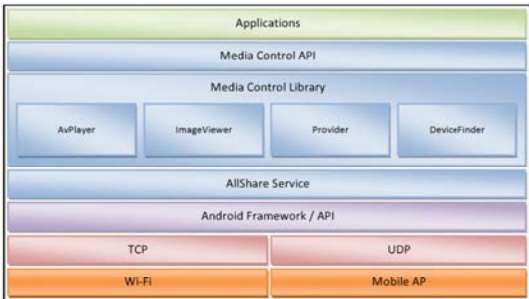


Figure 1: Media Control architecture
(See Samsung’s Media Control Programming Guide Version 1.0.1 at 11)

As shown below, Samsung’s Media Control Framework include a mechanism (e.g., SmcDeviceFinder) for discovering WLAN Devices, identifying Network Services provided by WLAN Devices, and listing the discovered device names and device types.

To discover devices that are part of the Media Control Framework:

1. Create an `SmcDeviceFinder` instance to retrieve `SmcDevice` instances from the Media Control Framework service. The `SmcDeviceFinder` class offers the following methods.

Method	Description
<code>getDevice(int deviceType, String id)</code>	Return the device with the specified device ID and device type.
<code>getDeviceList(int deviceType)</code>	Return the list of discovered devices with the specified device type.
<code>getDeviceList(int deviceType, String networkInterface)</code>	Return the list of discovered devices with the specified device type within the specified network interface controller.

2. Create a `SmcDeviceFinder.DeviceListener` listener and register it with your `SmcDeviceFinder` instance to receive device availability events. The `onDeviceAdded()` or `onDeviceRemoved()` methods are called when a device is added or removed in the Media Control network.

You can use the `rescan()` method to refresh the existing device list. Media Control sends a broadcast message in the network and the result is reported through the `SmcDeviceFinder.DeviceListener` listener.

(See Samsung’s Media Control Programming Guide Version 1.0.1 at 11)

As shown below, the Media Control Framework also identifies whether the WLAN Device provides a Network Service, such as a remote audio and video player service, a remote image viewer service, and/or a remote content provider Network Service.

Media Control provides several Media Control device types. After getting an `SmcDevice` instance, you can type cast it by device type.

Class	Device Type Enum	Description
<code>SmcAvPlayer</code>	<code>TYPE_AVPLAYER</code>	Remote Audio and Video Player Device
<code>SmcImageViewer</code>	<code>TYPE_IMAGEVIEWER</code>	Remote Image Viewer Device
<code>SmcProvider</code>	<code>TYPE_PROVIDER</code>	Remote Content Provider Device

(See Samsung’s Media Control Programming Guide Version 1.0.1 at 11)

Method	Description	Return Value
getDeviceDomain()	Return the device network domain.	int Example: DEVICE_DOMAIN_LOCAL_NETWORK
getDeviceType()	Return the device type.	int Example: DEVICE_TYPE_IMAGEVIEWER
getIconUri()	Return the representative icon URI of the device.	Example: "http://128.202.198.47:17676/DeviceIcon"
getIconList()	Return the list of all icons of the device.	A list of SmIcon
getId()	Return the device unique ID.	string Example: "uuid:08f0d180-0002-1000-8823-00245491c0ab"
getModelName()	Return the device model name.	Example: "Samsung Smart TV"
getName()	Return the user specified device name.	Example: "[TV]Smart TV"
getNetworkInterface()	Return the network interface to which the device is connected.	Network adapter name string
getIpAddress()	Return the device IP address.	Example: "111.22.33.4"


(See Samsung’s Media Control Programming Guide Version 1.0.1 at 12)

You can use Media Control to:

- Play media files saved on your device on another device
- Search for media files on a DLNA media server and play them on another device
- Play media files saved on a web server on another device

Playing Media Files on Another Device

Media Control allows you to play files saved on your device on another device.




(See Samsung’s Developer Webpage, “What is Media Control”)^{xxii}

Q. What devices can be discovered with the Media Control package?

A. Practically any device that supports the DLNA Digital Media Renderer feature :

- Samsung Smart TVs from 2009 or later.
- Windows Media Player version 11 or later.
- Any DLNA certified product that has the Digital Media Renderer feature.

For a list of certified products, see <http://www.dlna.org/dlna-for-industry/digital-living/product-search> 

(See Samsung’s Developer Webpage, “What is Media Control”)^{xxiii}

Similarly, Samsung’s AllShare Framework, as shown below, includes a mechanism (e.g., DeviceFinder) for discovering WLAN Devices and identifying Network Services provided by such WLAN Devices.

This section defines AllShare APIs that would be used to develop convergence services such as media sharing and control sharing. The AllShare APIs provides the below features applicable to any type of devices including mobile, smart tv, pc, etc:

- Media Sharing
 - discover DLNA devices in a network.
 - share media contents (picture, audio, video, etc) to other DLNA devices in a network.
 - browse and search the media contents shared by DLNA devices in a network.
 - playback media contents on a DLNA device in a network.
- Control Sharing
 - discover Smart TV in a network.
 - control TV and TV web browser remotely like a remote controller, mouse and keyboard did.
 - control TV viewer in detail as an extension of ImageViewer (DMR).

(See Samsung's Web API Guide)^{xxiv}

For example, as shown below, the AllShare Framework identifies whether the WLAN Device provides a Network Service, such as a media provider service, an AV player service, an image viewer service, and/or a TV Browser service.

To discover AllShare devices in the local network, the application should get a *DeviceFinder* instance by calling *getServiceFinder()*. In the AllShare Framework, an AllShare device in the local network is represented as an instance of *Device* class, and there are several device types which have different functionalities: Media Provider, AV Player, Image Viewer, and TV Browser.

```

Main.java
private void showDeviceList()
{
    if (mServiceProvider == null)
        return;

    DeviceFinder mDeviceFinder = mServiceProvider.getServiceFinder();
    mDeviceFinder.setDeviceFinderEventListener(DeviceType.DEVICE_AVPLAYER, mDeviceDiscoveryListener);
    ArrayList<Device> mDeviceList = mDeviceFinder.getDevices(DeviceDomain.LOCAL_NETWORK,
    DeviceType.DEVICE_AVPLAYER);

    if (mDeviceList != null) {
        for (int i = 0; i < mDeviceList.size(); i++) {
            mText.append("AVPlayer: " + mDeviceList.get(i).getName() + " [" + mDeviceList.get(i).getIpAddress() + "] is
found" + "\n\n");
        }
    }
}
    
```

(See AllShare Framework: Developer's Guide, Version 2.0)

As shown below, the AllShare Framework provides information including the device type, name, and model name.

Get Device Information

After getting the device list, an application can get information for each device using Device. The AllShare Framework provides the following device information.

Attribute	Description	Return Value
id	A string value represents the unique ID of the device, for example: "uuid:{a set of string}"	e.g. "uuid:080fd100-0002-1000-8823-00245491c0a9"
deviceDomain	A string value defined in DeviceDomain enumeration.	Device DeviceDomain enumeration
deviceType	A string value defined in DeviceType enumeration.	Device DeviceType enumeration
iconArray	A sequence of Icon objects.	A list of Icon objects.
ipAddress	A string value specifies the device IPv4 address. IPv4 addresses are canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots, for example, 172.16.254.1. Each part represents a group of 8 bits (octet) of the address.	e.g. "192.168.1.100"
modelName	A string value specifies the device model name.	e.g. "Samsung Smart TV"
name	A string value specifies the device name.	e.g. "[TV]Smart TV"
nic	A string value specifies the NIC (network interface controller) name of the device.	Network adaptor name string

(See Samsung’s Web API Guide, “Get Device Information”)^{xxv}

The device type is a value defined in Device Type Enumeration and identifies Network Services provided by WLAN Devices, such as Audio and video player, image viewer, media provider, media receiver, and TV controller services. The DeviceFinder API identifies services provided by WLAN Devices.

Interface	Method/Attribute	Description
DeviceFinder	Device? getDevice(DeviceType deviceType, DeviceId id)	Provides a device with a specified device ID and device type.
	DeviceArray getDeviceList(DeviceType deviceType)	Provides a list of discovered devices with specified device types.
	DeviceArray getDeviceListByDomain(DeviceType deviceType, DeviceDomain domain)	Provides a list of discovered devices with specified device types within specified network domains.
	DeviceArray getDeviceListByNIC(DeviceType deviceType, DOMString nic)	Provides a list of discovered devices with specified types within specified network interface controllers (NICs).

Through the DeviceFinder instance, the AllShare application can discover six types of devices in the network, each type of device is listed in the table below:

Device Type Enumeration	Description
AVPLAYER	Audio and video player device.
IMAGEVIEWER	Image viewer device.
FILERECEIVER	File receiver device.
MEDIAPROVIDER	Media provider device.
MEDIARECEIVER	Media receiver device.
TVCONTROLLER	TV controller device.

(See Samsung’s Web API Guide, “Get Device Information”)^{xxvi}

Samsung’s Web API (shown below) also identifies WLAN Devices and Network Services that they provide.

Samsung Web API



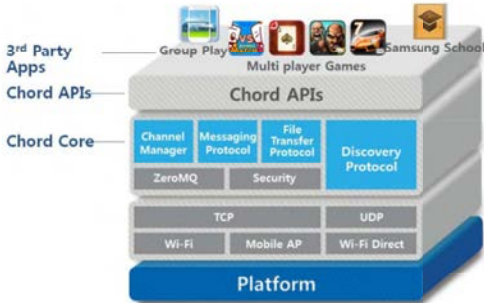
Samsung Web API allows developing web applications which are capable of controlling device features of Samsung devices. Samsung Web APIs and standard web technologies provide specialized features for creating highly qualified convergence web applications. It helps your application connect easily to various Samsung devices such as smart TV and mobile devices.

Download (Windows 7 32bit)	Download (Windows 7 64bit)
Samsung Web API Guide	Getting Started

(See Samsung’s Developer Webpage)^{xxvii}

Samsung Chord, which is supported by Samsung Relevant Devices with Android 4.0 or later, also includes service repository software components for identifying Network Services provided by WLAN Devices. (See Samsung’s Developer Webpage, “Samsung Chord”)^{xxviii} As shown below, the Chord architecture includes a discovery protocol.

The following figure shows the Chord architecture.



(See Chord Programming Guide Version 1.5)

Samsung Chord identifies, as shown below, Network Services provided by WLAN Devices, such as multiplayer

games, content-sharing, messaging, etc.

The Samsung Chord is a brilliant and fun way to share content and user events in real-time between devices. The Samsung Chord (Chord) SDK allows application developers to develop local information-sharing applications without a detailed knowledge of networking.

Chord enables simple real-time sharing without the cloud!

- Easy discovery and connection with nearby devices
- Real-time experiences between multiple devices
- Fast peer-to-peer connection, without the need of a server
- Decentralized networks where peers can come and go at any time

Chord quickly connects nearby devices so you can create shared, real-time experiences like collaborative interaction, multi-player games and media sharing. Without using the cloud or server, it instantly supports sharing 1-to-1, 1-to-many or many-to-many.

Devices running Chord-based applications discover each other using a UDP broadcast based discovery, and then use a TCP-based protocol stack to create a reliable, peer-to-peer local communication network. This network can be used to share data, including text, binary messages and files, with selected members of the network.

Chord SDK helps you create a group with multi-devices in real-time, automatically, requiring no manual processing of devices which join or leave the group.

(See Samsung’s Developer Webpage, “Samsung Chord”)^{xxx}

Samsung’s Accessory Service Framework, which is installed on Samsung Relevant Devices with Android 4.3 or later, also includes service repository software components for identifying Network Services provided by WLAN Devices.⁹ (See Samsung’s Developer Webpage, “Restrictions”)^{xxx}

As shown below, Accessory allows Samsung Relevant Devices (e.g., Smart Device) and WLAN Devices (e.g., Car head unit, gaming console, wireless speakers, printing devices, health care devices) to identify and share various Network Services.

⁹ Samsung’s webpage states that the only WLAN Devices that presently support Accessory are the Galaxy Gear, Gear 2, and Gear S (from January 2014 version).

What Is Accessory?

Accessory allows you to connect accessory devices to Samsung smart devices. With Accessory, you can define a new service between the accessory and smart device, and then use the various smart device functions from the accessory device. The service is compatible with various connectivity environments, and it makes accessory development efficient and convenient.

Accessory adds new functions to the service as Samsung smart devices improve. Future updates will enable the accessory and smart device to exchange more information and support more interworking.



Figure 1: Samsung Smart Device and Accessories
(See Samsung's Developer Webpage, "What is Accessory?")^{xxxii}

The following figure shows the roles in the Accessory eco-system.

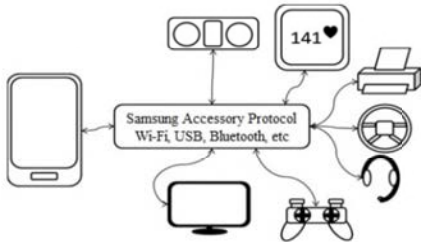


Figure 1: Accessory eco-system

Samsung Smart Devices can support one or more Accessory Services using a manager application with the Samsung Accessory Service Framework, for example, Samsung GEAR Manager. The Smart Devices and Accessory Devices described in this document have the Samsung Accessory Service Framework preloaded.

(See Accessory Programming Guide, Version 2.1.11)

Accessory also includes, as shown below, a mechanism to identify services and establish service connections.

The Samsung Accessory Protocol supports multiple connectivity methods, such as Wi-Fi, Bluetooth classic, Bluetooth Low Energy (v4.0), and USB, while freeing you from connectivity-specific details. The Samsung Accessory Service Framework supports the discovery of features (services) and enables the establishment of Service Connections between ALEs for data exchange.

(See Accessory Programming Guide, Version 2.1.11)

As shown below, Samsung Relevant Devices and WLAN Devices can identify and exchange Network Services using the Accessory Framework.

The following figure shows the functional flow in the Samsung Accessory Service Framework between a Service Consumer and a Service Provider. Peer Device 1 and 2 are either Samsung Smart Devices or Accessory Devices with applications acting as Service Providers or Service Consumers.

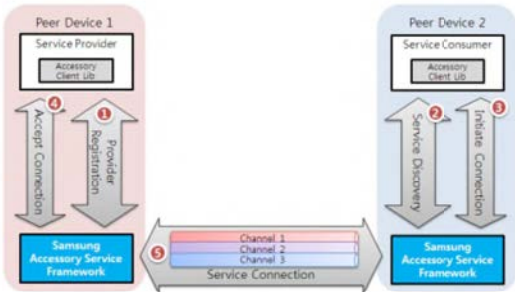


Figure 3: Functional flow between Service Provider and Service Consumer

(See Accessory Programming Guide, Version 2.1.11)

Accessory includes the Accessory Peer Agent, which identifies the available Network Services between peer devices.

1.5. Supported Features

Samsung works with domain experts within and outside Samsung to define Accessory Service Profiles. The Accessory Service Profiles define the application-level state machine and application-level protocol to implement domain-specific functionality. For example, the Notification Accessory Service Profile defines an application-level protocol to convey phone notifications to connected Accessory Devices.

Accessory supports the following features:

- Accessory Peer Agent:
 - Getting the list of Accessory Peer Devices.
 - Getting the list of services offered by the Accessory Peer Devices.
 - Identifying the available services between Peer Devices.

(See Accessory Programming Guide, Version 2.1.11)

Samsung Relevant Devices also include Bluetooth APIs for discovering Network Services. *See* Samsung's Answers to IXI's First Set of Requests for Admission, No. 30 ("Samsung admits that Samsung devices with Bluetooth capability support Bluetooth SDP."). For example, Samsung Relevant Devices include the Service Discovery Application and Profile as shown below.

Service Discovery Application and Profile: This profile defines the features and procedures for an application in a Bluetooth device to discover services registered in other Bluetooth devices, and retrieves information related to the services.

(See Samsung API Guide JSR 82- Bluetooth, Version .9 at 6)

Samsung Relevant Devices also include, for example, the `discoverDevices` API as shown below.

`discoverDevices`

Discovers nearby Bluetooth devices if any, that is, devices within proximity to the local device.

SIGNATURE

Method Signature

```
void discoverDevices(BluetoothDiscoverDevicesSuccessCallback successCallback, optional ErrorCallback? errorCallback);
```

This method initiates the device discovery process. Depending on the progress of this process the following methods are invoked:

- `BluetoothDiscoverDevicesSuccessCallback.onstarted()` - when a discovery process starts successfully.
- `BluetoothDiscoverDevicesSuccessCallback.ondevicefound()` - when any device is found in the process and this method is invoked with the device information. If no device is found, this method will never be invoked.
- `BluetoothDiscoverDevicesSuccessCallback.ondeviceisappeared()` - when a device goes out of proximity and this method is invoked with the address of the device.
- `BluetoothDiscoverDevicesSuccessCallback.onfinished()` - when a discovery process is completed.

(See Samsung's Web API Guide, "`discoverDevices`")^{xxxii}

The Bluetooth APIs allow Samsung Relevant Devices to identify and use Network Services provided by WLAN Devices and vice versa, for example as shown below.

Use Cases



Heart Rate Profile(HRP)



Proximity Profile(PXP)



Alert Notification Profile(ANP)

Profiles are high level definitions that define how services can be used to enable an application or use case. For further information on these profiles, visit : <https://www.bluetooth.org/Technical/Specifications/adopted.htm>

(See Samsung’s Developer Webpage, “Samsung BLE SDK”)^{xxxiii}

Operating systems of Samsung Relevant Devices may also include other service repository software components, such as APIs or Frameworks relating to Wi-Fi Direct (P2P), Universal Plug and Play (UPnP), Digital Living Network Alliance (DLNA), and/or reasonably similar protocols, which also provide mechanisms to identify Network Services provided by WLAN Devices.

As shown below, DLNA uses the UPnP Device Architecture for service discovery and identification.

Table 1 - Key Technology Ingredients

Functional Components	Technology Ingredients
Connectivity	Ethernet, 602.11 (including Wi-Fi Direct), MoCA, HD-PLC, HomePlug-AV, HPNA and Bluetooth
Networking	IPv4 Suite
Device Discovery and Control	UPnP [®] Device Architecture
Media Management and Control	UPnP AV, EnergyManagement, DeviceManagement, and Printer
Media Formats	Required and Optional Format Profiles
Media Transport	HTTP (Mandatory) , HTTP Adaptive Delivery (DASH) and RTP
Remote User Interfaces	CEA-2014-A , HTML5
Device Profiles	CVP-NA-1, CVP-EU-1, CVP-2

(See DLNA webpage, "Architectures and Protocols")^{xxxiv}

The UPnP Device architecture provides a mechanism to identify services provided by WLAN Devices and establish service connections, as shown below.

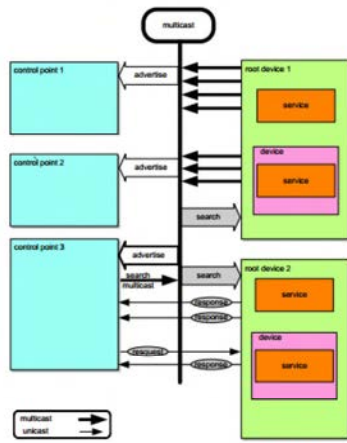
1 Discovery

Discovery is Step 1 in UPnP™ networking. Discovery comes after addressing (Step 0) where devices get a network address. Through discovery, control points find interesting device(s). Discovery enables description (Step 2) where control points learn about device capabilities, control (Step 3) where a control point sends commands to device(s), eventing (Step 4) where control points listen to state changes in device(s), and presentation (Step 5) where control points display a user interface for device(s).

Discovery is the first step in UPnP networking. When a device is added to the network, the UPnP discovery protocol allows that device to advertise its services to control points on the network. Similarly, when a control point is added to the network, the UPnP discovery protocol allows that control point to search for devices of interest on the network. The fundamental exchange in both cases is a discovery message containing a few, essential specifics about the device or one of its services, e.g., its type, universally unique identifier, a pointer to more detailed information and optionally parameters that identify the current state of the device.

(See UPnP Device Architecture 2.0, Sept 1, 2014 at 18)

Figure 1-1: — Discovery architecture



(See UPnP Device Architecture 2.0, Sept 1, 2014 at 19)

	<p>When a device knows it is newly added to the network, it shall multicast a number of discovery messages advertising itself, its embedded devices, and its services (initial announce). Any interested control point can listen to the standard multicast address for notifications that new capabilities are available. A multi-homed device shall multicast the discovery messages on all UPnP-enabled interfaces. A multi-homed control point is allowed to listen to the standard multicast address on one, some or all of its UPnP-enabled interfaces.</p> <p>When a new control point is added to the network, it is allowed to multicast a discovery message searching for interesting devices, services, or both. All devices shall listen to the standard multicast address for these messages and shall respond if any of their root devices, embedded devices or services matches the search criteria in the discovery message. In addition, a control point is allowed to unicast a discovery message to a specific IP address on port 1900 or on the port specified by the optional SEARCHPORT.UPnP.ORG header field (which supersedes port 1900 for this use), searching for a UPnP device or service at that specific IP address. This action presumes the control point already knows the device at this IP address is a UPnP device (which listens on the appropriate port). The control point can use unicast search for a number of applications. A unicast search can quickly confirm a specific device and provide the corresponding discovery information (e.g. UUID, URL) of this device.</p> <p style="text-align: right;">(UPnP Device Architecture 2.0, Sept 1, 2014 at 19)</p> <p>be different). The field value of the NEXTBOOTID.UPNP.ORG header field indicates the field value of the BOOTID.UPNP.ORG header field that a multi-homed device intends to use in future announcements after adding a new UPnP-enabled interface. The field value of the CONFIGID.UPNP.ORG header field identifies the current set of device and service descriptions; control points can parse this header field to detect whether they need to send new <i>description</i> query messages. The field value of the SEARCHPORT.UPNP.ORG header field identifies the port at which the device listens to unicast M-SEARCH messages; control points can parse this header field to know to which port unicast M-SEARCH messages shall be sent. These header fields are explained in detail below.</p> <p style="text-align: right;">(UPnP Device Architecture 2.0, Sept 1, 2014 at 21)</p> <p>1.2.2 Device available - NOTIFY with ssdp:alive</p> <p>When a device is added to the network, it shall multicast discovery messages to advertise its root device, any embedded devices, and any services. Each discovery message shall contain four major components:</p> <ol style="list-style-type: none"> a) A notification type (e.g., device type), sent in an NT (Notification Type) header field. b) A composite identifier for the advertisement, sent in a USN (Unique Service Name) header field. c) A URL for more information about the device (or enclosing device in the case of a service), sent in a LOCATION header field. d) A duration for which the advertisement is valid, sent in a CACHE-CONTROL header field. <p>To advertise its capabilities, a device multicasts a number of discovery messages. Specifically, a root device shall multicast:</p> <p style="text-align: right;">(See UPnP Device Architecture 2.0, Sept 1, 2014 at 24)</p>
--	--

Table 1-3 – Service discovery messages

	NT	USH #
1	<code>urn:schemas-upnp-org:device:serviceType:ver1</code> or <code>urn:domain-name:service:serviceType:ver</code>	<code>uuid:device-UUID:urn:schemas-upnp-org:device:serviceType:ver1</code> or <code>uuid:device-UUID:urn:domain-name:service:serviceType:ver</code>

^a Note that the field value of this NT header field shall match the value of the UDN element in the device description.

If a root device has *d* embedded devices and *s* embedded services but only *k* distinct service types, this works out to $3+2d+k$ requests. If a particular device or embedded device contains multiple instances of a particular service type, it is only necessary to advertise the service type once (rather than once for each instance). Note that if two embedded devices contain a service of the same service type, these services shall still be separately announced. This advertises the full extent of the device's capabilities to interested control points. These messages shall be sent out as a series with roughly comparable expiration times; order is unimportant, but refreshing or canceling individual messages is PROHIBITED.

Updated UPnP device and service types are required to be fully backward compatible with previous versions of the same type. Devices shall advertise the highest supported version of each supported type. For example, if a device supports version 2 of the "Audio" service, it would advertise only version 2, even though it also supports version 1. It shall NOT advertise additional supported versions. Control points that support a given version of a device or service are able to also interact with higher versions because of this backward compatibility requirement, but only using the functionality that was defined in the lower version. For example, if a control point supports only version "1" of the "Audio" service, and a device advertises that it supports version "2" of the "Audio" service, the control point shall recognize the device and be able to use it.

Choosing an appropriate duration for advertisements is a balance between minimizing network traffic and maximizing freshness of device status. Relatively short durations close to the minimum of 1800 seconds will ensure that control points have current device status at the expense of additional network traffic; longer durations, say on the order of a day, compromise freshness of device status but can significantly reduce network traffic. Generally, device vendors should choose a value that corresponds to expected device usage: short durations for devices that are expected to be part of the network for short periods of time, and significantly longer durations for devices expected to be long-term members of the network. Devices that frequently connect to and leave the network (such as mobile wireless devices) should use a shorter duration so that control points have a more accurate view of their availability. Advertisements in a set (both initial and subsequent) should have comparable durations. Advertisements in the initial set should be sent as quickly as possible. Subsequent refreshments of the advertisements are allowed to be spread over time rather than being sent as a group.

Spreading refreshments of advertisements over time rather than being sent as a group improves reliability in case there are network glitches; without increasing the total network load it increases the frequency of sending announcements from devices to control points. The two figures below show the announcement behavior without spreading and with spreading the

(See UPnP Device Architecture 2.0, Sept 1, 2014 at 25)

In addition or alternatively, applications (including Samsung or third party applications provided with Samsung Relevant Devices and/or downloaded onto Samsung Relevant Devices) include service repository software components. For example, applications compliant with the Android MediaRouter and MediaRouteProvider framework and APIs identify Network Services such as displaying and playback services provided by WLAN Devices as shown below.

Media route selector
When a user presses the Cast button, the media router framework looks for available media routes and presents a list of choices to the user, as shown in figure 3.

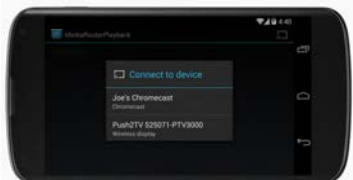


Figure 3. A list of available media routes, shown after pressing the Cast button.

(See Android's Developer Webpage, "MediaRouter"^{xxxv})



Figure 2. A Cast button shown on the right side of the action bar.

(See Android's Developer Webpage, "MediaRouter"^{xxxvi})

As another example, the Samsung Link application (formerly known as AllShare Play) includes DLNA functionality that allows for service discovery including, for example, by identifying whether the WLAN Device includes a Photo sharing service, a music sharing service, a video sharing service, and/or a document sharing service (as shown below).

- 3 Select the device or storage service containing the desired content.
Select the desired content to play from Photos, Music, Videos, Documents and Files tabs, and the selected content will be launched.
- Quality of file playback may vary depending on network environment or file size.



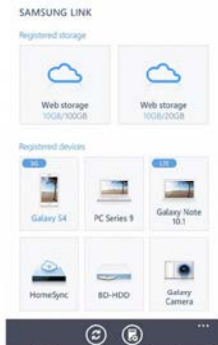
(See Samsung's Webpage, "Samsung Link")^{xxxvii}

**Share & play content
across smart devices
anywhere anytime**

Enjoy easy access to
all your content
- anywhere anytime -
with Samsung Link.



(See Samsung's Webpage, "Samsung Link")^{xxxviii}



In addition, as shown below, Samsung Link identifies when displaying services, such as the use of a large TV screen, are available.

Change player to large TV screen

Content on your mobile device or PC can be viewed from a large TV screen connected to the same network.

Mobile

1 Start Samsung Link. Press the select icon in the top right of the screen and select all files to send. When the Change Player icon in the top right of the screen is selected, content can be played on the TV connected to the same network.



(See Samsung’s Webpage, “Samsung Link”)^{xxxix}



Larger and clearer viewing with Samsung SMART TV

Move away from small screens!
Enjoy photos and videos taken from your smart phone and enjoy them with the whole family from the comfort of your Samsung SMART TV screen.
Content from devices and storage services registered in Samsung Link can be enjoyed wirelessly from a large TV screen without the hassle of complex connections or cables. Samsung Link is the single solution to all your needs.

(See Samsung’s Webpage, “Samsung Link”)^{xi}

In addition, AllShare Play identifies screen and media sharing network services that are provided by WLAN Devices.

Share it on the big screen

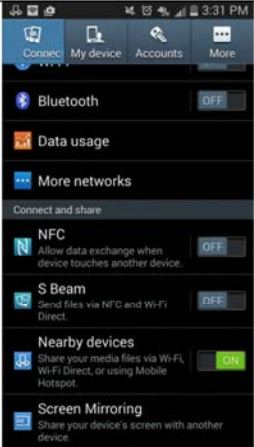
Instant Play


See the photos on your Smart Phone come to life on the big screen. With Instant Play, you can push content from your Samsung mobile devices right to your Smart TV™. Share your favorite videos, photos, and music with friends and family.



(See Samsung’s Webpage, “allshare play Share & Play Together”)^{xli}

As another example, Samsung Relevant Devices include the “Nearby Devices” feature, which uses DLNA to identify Network Services provided by WLAN Devices via Wi-Fi, Wi-Fi Direct, or Mobile Hotspot.

	 <p>Once Nearby Devices (or a reasonably similar feature) is activated, Samsung Relevant Devices identify available Network Services provided by WLAN Devices, for example, as shown below.</p>
--	--



Play music on DLNA-enabled devices.

Adjust the volume.

Set the file as your favourite song.

Turn on shuffle.

Change the repeat mode.

Hide the music player screen.

Restart the currently-playing song or skip to the previous song. Tap and hold to move backwards quickly.

Open the playlist.


Skip to the next song. Tap and hold to move forwards quickly.

Pause and resume playback.

(See Samsung Galaxy S4 User Manual at 75)

Playing videos

Select a video to play.



Scan DLNA-enabled devices.

Adjust the volume.

Move forwards or backwards by dragging the bar.

Skip to the next video. Tap and hold to move forwards quickly.

Change screen ratio.

Reduce the size of the video screen.

Restart the current video or skip to the previous video. Tap and hold to move backwards quickly.

Pause and resume playback.

(See Samsung Galaxy S4 User Manual at 75)

The Samsung Group Play application also includes a service repository component that identifies when gaming or other Network Services are provided by WLAN Devices. As shown below, Group Play uses Mobile Hotspot connections to create a WLAN.

Host your own games with Group Play

Hosting games for your friends is easy with Group Play. Once you launch the app, players can either create a group or join an existing one. You don't even need an internet connection to enjoy Group Play. Group Play uses Mobile Hotspots so that everyone can connect, even without an internet connection.



(See Samsung's Webpage, "Group Play")^{xiii}

Samsung Relevant Devices may also include other applications with service repository software components that identify Network Services provided by WLAN Devices as described in various protocols and APIs such as Network Service Discovery (NSD), Bluetooth Service Discovery Protocol, AllShare Framework, Media Control Framework, Chord Framework, Accessory Service Framework, Universal Plug and Play (UPnP), DLNA, and/or reasonably similar protocol or framework.

In addition, as shown below, the LTE Mobile Hotspot PRO includes DLNA functionality, which identifies services provided by WLAN via DLNA/UPnP service discovery.

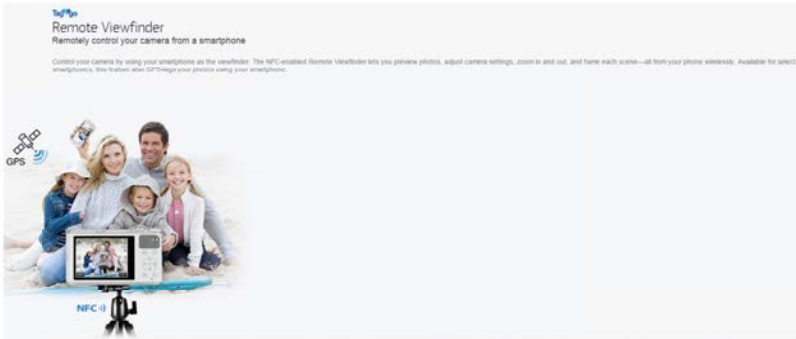
DLNA Tab


This tab provides you with the ability to designate your HotSpot or a DLNA-compliant device that can share stored information with other compliant devices.

This feature works by sharing data stored within an internally installed microSD memory card. For more information, refer to "Installing the Optional microSD Memory Card" on page 9.

(See Samsung LTE Mobile Hotspot PRO User Manual at 32)

Claim 4	
<p>The system of claim 1, wherein the service repository software component identifies whether the service is available at a particular time.</p>	<p>The content corresponding to Claim 1 is hereby incorporated by reference. As explained in the “service repository software component” of Claim 1, Samsung Relevant Devices identify Network Services as they become available.</p>
Claim 5	
<p>The system of claim 1, wherein the software component includes a domain naming service (“DNS”) software component to translate between a human readable name and a second Internet Protocol (“IP”) address.</p>	<p>The content corresponding to Claim 1 is hereby incorporated by reference. The Samsung Relevant Devices each have a software component (e.g., Wireless Hotspot Features and Android or Windows operating system software) that includes a DNS component to translate between a human readable name and a second IP address. For example, as shown below, the LTE Mobile Hotspot PRO includes a DNS setting.</p> <p style="text-align: center;">Note: DNS (Domain Name Server) should always be enabled when using Port Filtering. Without DNS, it is not possible to convert a Domain name (for example, www.msn.com) to an IP address. DNS is used by E-mail, FTP, and many other protocols as well as by Web browsers.</p> <p style="text-align: right;">(See Samsung LTE Mobile Hotspot PRO User Manual at 32)</p> <p>Android APIs also include InetAddress, getHostname, and C Library which translate between a human readable hostname and an IP address.</p> <p>DNS caching</p> <p><small>In Android 4.0 (Ice Cream Sandwich) and earlier, DNS caching was performed both by InetAddress and by the C library, which meant that DNS TTLs could not be honored correctly. In later releases, caching is done solely by the C library and DNS TTLs are honored.</small></p> <p style="text-align: right;">(See Android Developer Webpage, “InetAddress”)^{xliii}</p>
Claim 6	
<p>The system of claim 1, wherein the software component includes a security software component to control access between the cellular network and the</p>	<p>The content corresponding to Claim 1 is hereby incorporated by reference. Samsung Relevant Devices include security software components that allow the Samsung Relevant Devices to control access between the cellular network and the first wireless device. For example, Samsung Relevant Devices include software that utilizes MAC filtering, port filtering, VPNs (Virtual Private Networks), Bluetooth PINs, and/or adheres to security protocols such as Wi-Fi Protected Access (WPA) and/or WPA II. (Samsung Galaxy S4 User Manual at 121)</p>

first wireless device.	Claim 7
The system of claim 1, wherein the second wireless device is a thin terminal.	<p>The content corresponding to Claim 1 is hereby incorporated by reference. A WLAN Device is the claimed second wireless device. A WLAN Device may be a thin terminal such as a printer or camera. For example, Samsung Relevant Devices can communicate wirelessly with printers via applications such as the Samsung Print Service Plugin. (Samsung Galaxy S4 User Manual at 99) Samsung Relevant Devices can also communicate wirelessly with smart cameras via applications and features such as the Samsung Smart Camera App, AllShare Play, Remote Viewfinder, and MobileLink.</p> <div data-bbox="500 850 1291 1186" style="border: 1px solid #ccc; padding: 10px; text-align: center;"><p>Remote Viewfinder Remotely control your camera from a smartphone</p><p><small>Control your camera by using your smartphone as the viewfinder. The NFC-enabled Remote Viewfinder lets you preview photos, adjust camera settings, zoom in and out, and frame each scene—all from your phone wirelessly. Available for select smartphones. This feature uses GPS to tag your photos using your smartphone.</small></p></div> <p style="text-align: right;">(See Samsung's Webpage, "Remote Viewfinder")^{xliv}</p>

	 <p style="text-align: center;">(See Samsung Smart Camera YouTube Advertisement)^{xiv}</p>
Claim 12	
<p>The system of claim 1, wherein the software component includes a plug and play software component to load and execute software for the second wireless device.</p>	<p>The content corresponding to Claim 1 is hereby incorporated by reference. Samsung Relevant Devices' software components include plug and play software components. For example, Samsung Relevant Devices include UPnP and/or DLNA APIs or reasonably similar APIs that may be used to load and execute software for WLAN Devices that are compatible with DLNA and/or UPnP. In addition, Samsung Relevant Devices and WLAN Devices include AllShare Play/Link, Samsung Print Services, and/or Samsung PC Share Manager software components that are DLNA or UPnP certified and offer plug and play functionality.</p>
Claim 13	
<p>The system of claim 1, wherein the software component includes a PIN number management software component to obtain and provide PIN numbers.</p>	<p>The content corresponding to Claim 1 is hereby incorporated by reference. Samsung Relevant Devices software components include PIN number management software to obtain and provide PIN numbers. For example, Bluetooth Tethering requires a pairing PIN Code.</p> <p style="color: red;">Pairing with other Bluetooth devices</p> <p>On the Applications screen, tap Settings → Connections → Bluetooth → Scan, and detected devices are listed. Select the device you want to pair with, and then accept the auto-generated passkey on both devices to confirm.</p> <p style="text-align: right;">(See Samsung Galaxy S4 User Manual at 68)</p>
Claim 14	
<p>The system of claim 1, wherein the second</p>	<p>The content corresponding to Claim 1 is hereby incorporated by reference. As described for the "service repository software component" of Claim 1, a WLAN Device is the claimed second wireless device and includes Network</p>

wireless device includes an application software component that registers an availability of the service with the service repository software component.

Service applications or software that are identified by the service repository software component on Samsung Relevant Devices.

Network Service software registers availability of Network Services with software components of Samsung Relevant Devices, such as operating system software (e.g., Android or Windows), Wireless Hotspot Feature software, and/or other Network Service software, for example, via the various APIs, frameworks, and protocols discussed in relation to the “service repository software component” of Claim 1.

For example, the availability of Network Services is registered with the service repository software component via the Android NSD APIs, as shown below.

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.

```
public void registerService(int port) {
    // Create the NsdServiceInfo object, and populate it.
    NsdServiceInfo serviceInfo = new NsdServiceInfo();

    // The name is subject to change based on conflicts
    // with other services advertised on the same network.
    serviceInfo.setServiceName("NsdChat");
    serviceInfo.setServiceType("_http_tcp.");
    serviceInfo.setPort(port);
    ....
}
```

This code snippet sets the service name to "NsdChat". 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.

```
public void initializeServerSocket() {
    // Initialize a server socket on the next available port.
    mServerSocket = new ServerSocket(0);

    // Store the chosen port.
    mLocalPort = mServerSocket.getLocalPort();
    ...
}
```

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()`.

(See Android Developer Webpage, "Using Network Service Discovery")^{xlvi}

Similarly, availability of Network Services is registered with the service repository software via the Android `MediaRouter` and `MediaRouteProvider` frameworks and APIs, for example, as shown below.

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`.

(See Android's Developer Webpage, "Media Router Provider")^{xlvii}

Media route providers are used to publish additional media routes for use within an application. Media route providers may also be declared as a service to publish additional media routes to all applications in the system.

The purpose of a media route provider is to discover media routes that satisfy the criteria specified by the current `MediaRouteDiscoveryRequest` and publish a `MediaRouteProviderDescriptor` with information about each route by calling `setDescriptor(MediaRouteProviderDescriptor)` to notify the currently registered `MediaRouteProvider.Callback`.

The provider should watch for changes to the discovery request by implementing `onDiscoveryRequestChanged(MediaRouteDiscoveryRequest)` and updating the set of routes that it is attempting to discover. It should also handle route control requests such as volume changes or media control intents by implementing `onCreateRouteController(String)` to return a `MediaRouteProvider.RouteController` for a particular route.

A media route provider may be used privately within the scope of a single application process by calling `MediaRouter.addProvider` to add it to the local `MediaRouter`. A media route provider may also be made available globally to all applications by registering a `MediaRouteProviderService` in the provider's manifest. When the media route provider is registered as a service, all applications that use the media router API will be able to discover and use the provider's routes without having to install anything else.

(See Android's Developer Webpage, "MediaRouteProvider"^{xlviii})

Similarly, availability of Network Services is registered with the service repository software via the Android Wi-Fi P2P APIs, as shown below.

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.

(See Android Developer Webpage, "Using Wi-Fi P2P for Service Discovery"^{xlix})

The `WifiP2pManager` class provides methods to allow you to interact with the Wi-Fi hardware on your device to do things like discover and connect to peers. The following actions are available:

Table 1. Wi-Fi P2P Methods

Method	Description
<code>initialize()</code>	Registers the application with the Wi-Fi framework. This must be called before calling any other Wi-Fi P2P method.
<code>connect()</code>	Starts a peer-to-peer connection with a device with the specified configuration.
<code>cancelConnect()</code>	Cancels any ongoing peer-to-peer group negotiation.
<code>requestConnectInfo()</code>	Requests a device's connection information.
<code>createGroup()</code>	Creates a peer-to-peer group with the current device as the group owner.
<code>removeGroup()</code>	Removes the current peer-to-peer group.
<code>requestGroupInfo()</code>	Requests peer-to-peer group information.
<code>discoverPeers()</code>	Initiates peer discovery
<code>requestPeers()</code>	Requests the current list of discovered peers.

`WifiP2pManager` methods let you pass in a listener, so that the Wi-Fi P2P framework can notify your activity of the status of a call. The available listener interfaces and the corresponding `WifiP2pManager` method calls that use the listeners are described in the following table:

(See Android Developer Webpage, "Wi-Fi Peer-to-peer")¹

As another example, availability of Network Service applications (Service Provide and Service Consumer applications below) is registered with the Samsung Accessory Service Framework service repository software component.

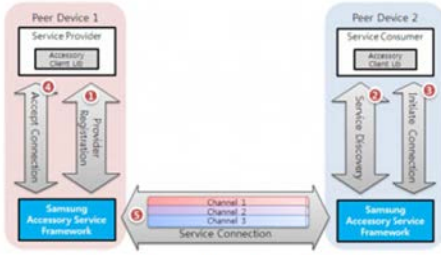


Figure 3: Functional flow between Service Provider and Service Consumer
The Service Provider and Service Consumer applications register their service capabilities with the Samsung Accessory Service Framework. The Samsung Accessory Service Framework advertises and exchanges the capabilities of the registered Service Providers and Service Consumers.

(See Accessory Programming Guide, Version 2.1.11 at 6)

Availability of Network Services is also registered with the service repository software via the Bluetooth APIs, for example as shown below.

Service Discovery Application and Profile: This profile defines the features and procedures for an application in a Bluetooth device to discover services registered in other Bluetooth devices, and retrieves information related to the services.

(See Samsung API Guide JSR 82- Bluetooth, Version .9 at 6)

```
BluetoothHealthCallback
An abstract class that you use to implement BluetoothHealth callbacks. You must extend this class and implement the callback methods to receive updates about changes in the application's registration state and Bluetooth channel state.

BluetoothHealthAppConfiguration
Represents an application configuration that the Bluetooth Health third-party application registers to communicate with a remote Bluetooth health device.
```

(See Android Developer Webpage, "Bluetooth Class")ⁱⁱ

In using the Bluetooth Health API, it's helpful to understand these key HDP concepts:

Concept	Description
Source	A role defined in HDP. A <i>source</i> is a health device that transmits medical data (weight scale, glucose meter, thermometer, etc.) to a smart device such as an Android phone or tablet.
Sink	A role defined in HDP. In HDP, a <i>sink</i> is the smart device that receives the medical data. In an Android HDP application, the sink is represented by a <code>BluetoothHealthAppConfiguration</code> object.
Registration	Refers to registering a sink for a particular health device.
Connection	Refers to opening a channel between a health device and a smart device such as an Android phone or tablet.

(See Android Developer Webpage, "Bluetooth Class")ⁱⁱⁱ

To create a profile all you need to do is get the proxy object using `getProfileProxy(Context, BluetoothProfile.ServiceListener, int)` using a `BluetoothGattAdapter` and then call the `register` app method once the service is connected. For example:

```
public void onCreate() {
    if (mBluetoothAdapter == null) {
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        if (mBluetoothAdapter == null) return;
    }
    BluetoothGattAdapter.getProfileProxy(this,
    mProfileServiceListener, BluetoothGattAdapter.GATT);
    .
    .
    .
    private BluetoothProfile.ServiceListener mProfileServiceListener =
    new BluetoothProfile.ServiceListener() {
        public void onServiceConnected(int profile, BluetoothProfile
    proxy) {
        if (profile == BluetoothGattAdapter.GATT) {
            mBluetoothGatt = (BluetoothGatt) proxy;
            mBluetoothGatt.registerApp(mGattCallbacks);
        }
    }
}
```

Once the app is registered, you can get a `BluetoothGattCallback` or `BluetoothGattServerCallback` based on the proxy obtained and proceed from there on.

(See Guide and Hints for Samsung BLE API at 8)

Claim 15

The system of claim 1, further comprising: a

The content corresponding to Claim 1 is hereby incorporated by reference. Android's Wireless Hotspot Feature, for example, allows up to 10 WLAN Devices to connect at one time to a single Samsung Relevant Device.

third wireless device, in the short distance wireless network, having an application software component to obtain the service from the second wireless device.

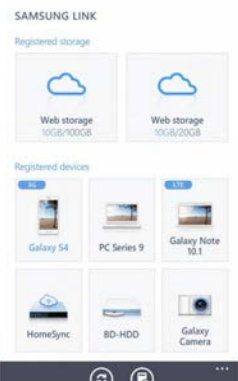
A WLAN Device is the claimed second wireless device. A second WLAN Device is the claimed third wireless device in the short distance network. The second WLAN Device includes Network Service software components so that it can obtain a Network Service from the first WLAN Device directly via short range radio signals (e.g., Bluetooth, Wi-Fi) or indirectly via a Samsung Relevant Device.

For example, as shown in “service repository software component” for Claim 1, a second WLAN Device can obtain Network Services from a first WLAN Device as described in various protocols and APIs. For example, Samsung Link (formerly known as AllShare Play) includes DLNA functionality that allows a second WLAN Device (e.g., a Galaxy Note tablet) to obtain Network Services from a first WLAN Device (e.g., an Office PC) such as a Photo sharing service, a music sharing service, a video sharing service, and/or a document sharing service (as shown below).

- 3 Select the device or storage service containing the desired content.
Select the desired content to play from Photos, Music, Videos, Documents and Files tabs, and the selected content will be launched.
• Quality of file playback may vary depending on network environment or file size.



(See Samsung's Webpage, “Share and Play Content across smart devices anywhere anytime”)^{liii}

	 <p>The screenshot shows the Samsung Link application interface. At the top, it says 'SAMSUNG LINK'. Below that, there are two sections: 'Registered storage' and 'Registered devices'. Under 'Registered storage', there are two icons for 'Web storage' with capacities of '16GB/100GB' and '16GB/20GB'. Under 'Registered devices', there are six icons representing different devices: 'Galaxy S4', 'PC Series 9', 'Galaxy Note 10.1', 'HomeSync', 'BD-HDD', and 'Galaxy Camera'. At the bottom of the interface, there are three circular icons: a refresh icon, a home icon, and a menu icon.</p>
<p>Claim 16</p>	
<p>The system of claim 15, wherein the first wireless device includes a service logical driver corresponding to the service, and wherein the application software component uses the service logical driver to obtain the service from the second wireless device.</p>	<p>The content corresponding to Claim 15 is hereby incorporated by reference. Samsung Relevant Devices include service logical drivers that correspond to Network Services and that are used by application software to obtain Network Services from WLAN Devices. A Samsung Relevant Device is the claimed first wireless device. A first WLAN Device is the claimed second wireless device, and a second WLAN Device is the claimed third wireless device. For example, as shown below, Samsung Relevant Devices include the Samsung Chord API which includes a service logical driver corresponding to a Network Service. Application software components use the service logical driver to obtain the Chord Network Service from WLAN Devices.</p> <p>Chord allows you to easily develop local information-sharing applications. Devices running Chord-based applications locate each other using UDP-broadcast-based discovery, and then use a TCP/IP-based protocol stack to create a reliable, local, peer-to-peer communications network. Chord-based applications use this network to share data, including text messages, binary messages and files, with selected members of the network. Chord version 2.0 and above also allows you to use UDP-based protocol stack to transfer time-sensitive data, including video, audio or game.</p> <p>You can use Chord to:</p> <ul style="list-style-type: none"> • Discover devices (nodes). • Join and leave private channels. • Share information by sending and receiving data and files. <p style="text-align: right;">(See Samsung’s Developer Webpage, “Chord – Programming Guide”)^{liv}</p>

Claim 22	
<p>The system of claim 1, wherein the service repository software component identifies a class, attribute and instance of the service.</p>	<p>The content corresponding to Claim 1 is hereby incorporated by reference. The service repository software identifies a class, attribute and instance of the service. For example, the Android Network Service Discovery API informs application when instances of services are found, the service name, and the service type or class.</p> <p>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.</p> <ol style="list-style-type: none"> 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. <p>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".</p> <p>After setting up the listener, call <code>discoverServices()</code>, passing in the service type your application should look for, the discovery protocol to use, and the listener you just created.</p> <p style="text-align: center;">(See Android Developer Webpage, "Using Network Service Discovery")^{lv}</p> <p>This code snippet sets the service name to "NsdChat". 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)".</p> <p>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".</p> <p>Note: The International Assigned Numbers Authority (IANA) manages a centralized, authoritative list of service types used by service discovery protocols such as NSD and Bonjour. You can download the list from the IANA list of service names and port numbers. If you intend to use a new service type, you should reserve it by filling out the IANA Ports and Service registration form.</p> <p style="text-align: center;">(See Android Developer Webpage, "Using Network Service Discovery")^{lvii}</p> <p>Android also includes Bluetooth APIs that allows compliant applications to "connect to other devices through service discovery." (See Android Developer Webpage, "Bluetooth")^{lviii} As shown below, the Android Bluetooth APIs identify services by class.</p>

	<p>Class Overview</p> <p>Represents a Bluetooth class, which describes general characteristics and capabilities of a device. For example, a Bluetooth class will specify the general device type such as a phone, a computer, or headset, and whether it's capable of services such as audio or telephony.</p> <p>Every Bluetooth class is composed of zero or more service classes, and exactly one device class. The device class is further broken down into major and minor device class components.</p> <p><code>BluetoothClass</code> is useful as a hint to roughly describe a device (for example to show an icon in the UI), but does not reliably describe which Bluetooth profiles or services are actually supported by a device. Accurate service discovery is done through SDP requests, which are automatically performed when creating an RFCOMM socket with <code>createRFCOMMSocketToServiceRecord(UUID)</code> and <code>listenRFCOMMWithServiceRecord(String, UUID)</code>.</p> <p>Use <code>getBluetoothClass()</code> to retrieve the class for a remote device.</p> <p>(See Android Developer Webpage, "Bluetooth Class")^{lviii}</p> <p>In addition, DLNA and UPnP allow compliant applications to identify services by type, attribute, and instance, for example, as shown below.</p>
--	--

Table 1-3 — Service discovery messages

	NT	UDN #
1	<pre> urn:ietf:params:xml:ns:serviceType:ver or urn:ietf:params:xml:ns:serviceType:ver </pre>	<pre> uid:device-UUID:urn:ietf:params:xml:ns:serviceType:ver or uid:device-UUID:urn:ietf:params:xml:ns:serviceType:ver </pre>

^a Note that the field value of this NT header field shall match the value of the UDN element in the device description.

If a root device has *d* embedded devices and *f* embedded services but only *k* distinct service types, this works out to $3 \times 2^{d+k}$ requests. If a particular device or embedded device contains multiple instances of a particular service type, it is only necessary to advertise the service type once (rather than once for each instance). Note that if two embedded devices contain a service of the same service type, these services shall still be separately announced. This advertises the full extent of the device's capabilities to interested control points. These messages shall be sent out as a series with roughly comparable expiration times; order is unimportant, but refreshing or canceling individual messages is PROHIBITED.

Updated UPnP device and service types are required to be fully backward compatible with previous versions of the same type. Devices shall advertise the highest supported version of each supported type. For example, if a device supports version 2 of the "Audio" service, it would advertise only version 2, even though it also supports version 1. It shall NOT advertise additional supported versions. Control points that support a given version of a device or service are able to also interact with higher versions because of this backward compatibility requirement, but only using the functionality that was defined in the lower version. For example, if a control point supports only version "1" of the "Audio" service, and a device advertises that it supports version "2" of the "Audio" service, the control point shall recognize the device and be able to use it.

Choosing an appropriate duration for advertisements is a balance between minimizing network traffic and maximizing freshness of device status. Relatively short durations close to the minimum of 1800 seconds will ensure that control points have current device status at the expense of additional network traffic; longer durations, say on the order of a day, compromise freshness of device status but can significantly reduce network traffic. Generally, device vendors should choose a value that corresponds to expected device usage: short durations for devices that are expected to be part of the network for short periods of time, and significantly longer durations for devices expected to be long-term members of the network. Devices that frequently connect to and leave the network (such as mobile wireless devices) should use a shorter duration so that control points have a more accurate view of their availability. Advertisements in a set (both initial and subsequent) should have comparable durations. Advertisements in the initial set should be sent as quickly as possible. Subsequent refreshments of the advertisements are allowed to be spread over time rather than being sent as a group.

Spreading refreshments of advertisements over time rather than being sent as a group improves reliability in case there are network glitches; without increasing the total network load it increases the frequency of sending announcements from devices to control points. The two figures below show the announcement behavior without spreading and with spreading the

(See UPnP Device Architecture 2.0, Sept 1, 2014 at 25)

Claim 23

The system of claim 1, wherein the first wireless device further includes a virtual private network ("VPN") software component.

The content corresponding to Claim 1 is hereby incorporated by reference. A Samsung Relevant Device is the claimed first wireless device. Samsung Relevant Devices include VPN software components. For example, the Samsung Galaxy S4 includes a VPN software component.

VPN


Set up and connect to virtual private networks (VPNs).

(See Samsung Galaxy S4 User Manual at 121)

Claim 24	
<p>The system of claim 1, wherein the first wireless device further includes a firewall software component.</p>	<p>The content corresponding to Claim 1 is hereby incorporated by reference. A Samsung Relevant Device is the claimed first wireless device. Samsung Relevant Devices include firewall software components. For example, the Samsung LTE Mobile HotSpot Pro includes a firewall software component as shown below.</p> <p style="text-align: center;">Port Forwarding Panel</p> <p style="text-align: center;">Port Forwarding allows incoming traffic (from the Internet) to be forwarded to a particular PC or device on your local WLAN. Normally, incoming traffic from the Internet is blocked by the Firewall.</p> <p style="text-align: center;">You need to use Port Forwarding to allow Internet users to access running services such as a Web server, FTP server, E-mail server, etc...</p> <p style="text-align: center;">For some online applications (such as games), Port Forwarding must be used in order for the game to function correctly.</p> <p style="text-align: right;">(See Samsung LTE Mobile Hotspot PRO User Manual at 35)</p> <p>In addition, the Linux kernel within Android includes the netfilter and/or iptables firewall.</p>
Claim 25	
<p>A system for providing access to information on a cellular network, comprising:</p>	<p>Although the preamble to Claim 25 does not limit the scope of the claim, Samsung provides systems and/or components of systems for providing access to information on a cellular network. Accused systems include Cellular Network Devices,¹⁰ Samsung Relevant Devices, and WLAN Devices.</p> <p>The Samsung Relevant Devices provide WLAN Devices with access to information on a cellular network (i.e., on Cellular Network Devices) via wireless radio signals (e.g., 802.11, Bluetooth) and cellular radio signals. WLAN Devices connect to Samsung Relevant Devices via short distance radio waves using the 802.11 and/or Bluetooth protocol and use the Samsung Relevant Device to access Cellular Network Devices.</p> <p>For example, all Samsung Relevant Devices include Wireless Hotspot Features. With the exception of the LTE Mobile Hotspot PRO and the SCH-LC11 4G Mobile Hotspot, all Samsung Relevant Devices include the Android Operating System version 2.2 or greater or the Windows Phone Operating System version 7 or greater. These operating system versions each include Wireless Hotspot Features. (See Wired Webpage, “Android 2.2 ‘Froyo’ Features USB, Wi-Fi Tethering”^{lix}; Windows Phone Webpage, “Share my Connection”^{lx})</p>

¹⁰ “Cellular Network Devices” are computers or other devices that function as application (e.g., Link server) or other cellular network servers, including those owned or operated by Samsung or third parties.

	<p>The Samsung Galaxy S4, for example, includes “Portable Wi-Fi hotspot” feature, which allows WLAN Devices to connect to the Samsung Galaxy S4 via Wi-Fi and use its cellular connection. (See Samsung Galaxy S4 User Manual at 121)</p> <p>Similarly, the Samsung ATIV S Neo includes an “Internet Sharing” feature, which allows WLAN Devices to connect to the Samsung ATIV S Neo via Wi-Fi and use its cellular connection. (See Samsung ATIV S Neo User Manual (Sprint) at 61)</p> <p>The LTE Mobile Hotspot PRO and the SCH-LC11 4G Mobile Hotspot are each themselves mobile hotspots with the primary purpose of providing Wireless Hotspot Features.</p>
<p>a first wireless device, in a short distance wireless network, to provide a first short-range radio signal; and,</p>	<p>A WLAN Device is the claimed first wireless device in a short distance wireless network (e.g., Wi-Fi 802.11 and/or Bluetooth network) that provides the first short-range radio signal to a Samsung Relevant Device. WLAN Devices connect to Samsung Relevant Devices via short distance radio waves using the 802.11 and/or Bluetooth protocol and use the Samsung Relevant Device to access Cellular Network Devices when using Wireless Hotspot Features or providing other Network Services via Wi-Fi Direct (P2P), Bluetooth, or reasonably similar protocols.</p>
<p>a second wireless device, in the short distance wireless network and the cellular network, to selectively transfer information, including Internet Protocol (“IP”) data packets, between the first wireless device and the cellular network in response to a security software component,</p>	<p>A Samsung Relevant Device is the claimed second wireless device in a short distance network and the cellular network. For example, when the Galaxy S4 is in Wireless Hotspot Mode it creates a short distance Bluetooth or Wi-Fi network and is also in a cellular (e.g., GSM, CDMA, or LTE) network. As another example, when the Galaxy S4 connects with a WLAN Device via Wi-Fi Direct or Bluetooth PAN, it is in both a short distance network and a cellular network.</p> <p>Samsung Relevant Devices include security software components that allow the Samsung Relevant Devices to selectively transfer information, including Internet Protocol (“IP”) data packets, between the WLAN Device and the Cellular Network Device in response to the security software receiving proper authorization. For example, Samsung Relevant Devices may include software (e.g., Wireless Hotspot Feature, Android, Microsoft Windows, or other Network Service applications) that utilizes port filtering, MAC filtering, VPNs (Virtual Private Networks), Bluetooth PINs, and/or adheres to security protocols such as Wi-Fi Protected Access (WPA) and/or WPA II. Wireless Hotspot Features also have an “allowed devices” feature that can be used to limit devices allowed to utilize the hotspot feature by MAC address. For example, as shown below, the Wireless Hotspot Features of the Samsung Galaxy S4 utilize WPA2 PSK.</p>

	 <p>Samsung Relevant Devices also include other security software that selectively transfers information based on proper authorization, for example, application or Network Service passwords such as a Samsung Link password.</p> <p>IP data packets are not transmitted from the Samsung Relevant Device to the Cellular Network Device unless the proper password is entered authorizing the WLAN Device to use the Wireless Hotspot Feature and/or other Network Service.</p>
<p>wherein the second wireless device includes a service repository software component that identifies a plurality of services, in the short distance wireless network, associated with a plurality of wireless devices, and</p>	<p>Samsung Relevant Devices include service repository software components for identifying a plurality of Network Services in the plurality of services provided by the WLAN Devices. The Network Services may be utilized by compatible Network Service application software stored in WLAN Devices.</p> <p>As explained for the Claim 1 “service repository software component,” which explanation is hereby incorporated by reference, Samsung Relevant Devices include a service repository software component for identifying a plurality of Network Services in the short distance network (e.g., Wi-Fi or Bluetooth network) associated with a plurality of WLAN Devices.</p>
<p>wherein the service repository software component searches for a service, in the plurality of services, to be used by an application software component stored in</p>	<p>The service repository software component also searches for a Network Service in the plurality of services to be used by a compatible Network Service application software component stored on a WLAN Device, for example, as described in the various APIs, frameworks, and protocols discussed in the “service repository software component” for Claim 1, which description is hereby incorporated by reference.</p> <p>For example, as shown below, Android NSD APIs provide a mechanism for searching for Network Services to be used by application software on a WLAN Device.</p>

the first wireless device.

Discover Services on the Network

The network is teeming with life, from the beastly network printers to the docile network webcams, to the brutal, fiery battles of nearby tic-tac-toe players. The key to letting your application see this vibrant ecosystem of functionality is service discovery. Your application needs to listen to service broadcasts on the network to see what services are available, and filter out anything the application can't work with.

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:

(See Android Developer Webpage, "Using Network Service Discovery")^{ixi}

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 `NsdManager.ResolveListener` to pass into this method, and use it to get a `NsdServiceInfo` containing the connection information.

```
public void initializeResolveListener() {
    mResolveListener = new NsdManager.ResolveListener() {

        @Override
        public void onResolveFailed(NsdServiceInfo serviceInfo, int errorCode) {
            // Called when the resolve fails. Use the error code to debug.
            Log.e(TAG, "Resolve failed" + errorCode);
        }

        @Override
        public void onServiceResolved(NsdServiceInfo serviceInfo) {
            Log.e(TAG, "Resolve Succeeded. " + serviceInfo);

            if (serviceInfo.getServiceName().equals(mServiceName)) {
                Log.d(TAG, "Same IP.");
                return;
            }
            mService = serviceInfo;
            int port = mService.getPort();
            InetAddress host = mService.getHost();
        }
    };
}
```

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.

(See Android Developer Webpage, "Using Network Service Discovery")^{lxii}

As another example, as shown below, Android MediaRouter and MediaRouteProvider framework and APIs provide a

mechanism for searching for Network Services to be used by application software on a WLAN Device.

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`.

(See Android's Developer Webpage, "Media Router Provider")^{lxiii}

Media route providers are used to publish additional media routes for use within an application. Media route providers may also be declared as a service to publish additional media routes to all applications in the system.

The purpose of a media route provider is to discover media routes that satisfy the criteria specified by the current `MediaRouteDiscoveryRequest` and publish a `MediaRouteProviderDescriptor` with information about each route by calling `setDescriptor(MediaRouteProviderDescriptor)` to notify the currently registered `MediaRouteProvider.Callback`.

The provider should watch for changes to the discovery request by implementing `onDiscoveryRequestChanged(MediaRouteDiscoveryRequest)` and updating the set of routes that it is attempting to discover. It should also handle route control requests such as volume changes or `media control intents` by implementing `onCreateRouteController(String)` to return a `MediaRouteProvider.RouteController` for a particular route.

A media route provider may be used privately within the scope of a single application process by calling `MediaRouter.addProvider` to add it to the local `MediaRouter`. A media route provider may also be made available globally to all applications by registering a `MediaRouteProviderService` in the provider's manifest. When the media route provider is registered as a service, all applications that use the media router API will be able to discover and use the provider's routes without having to install anything else.

(See Android's Developer Webpage, "MediaRouteProvider")^{lxiv}

As another example, as shown below, Wi-Fi P2P APIs provide a mechanism for searching for Network Services to be used by application software on a WLAN Device.

	<p>The <code>WifiP2pManager</code> class provides methods to allow you to interact with the Wi-Fi hardware on your device to do things like discover and connect to peers. The following actions are available:</p> <p>Table 1. Wi-Fi P2P Methods</p> <table border="1"><thead><tr><th>Method</th><th>Description</th></tr></thead><tbody><tr><td><code>initialize()</code></td><td>Registers the application with the Wi-Fi framework. This must be called before calling any other Wi-Fi P2P method.</td></tr><tr><td><code>connect()</code></td><td>Starts a peer-to-peer connection with a device with the specified configuration.</td></tr><tr><td><code>cancelConnect()</code></td><td> Cancels any ongoing peer-to-peer group negotiation.</td></tr><tr><td><code>requestConnectInfo()</code></td><td>Requests a device's connection information.</td></tr><tr><td><code>createGroup()</code></td><td>Creates a peer-to-peer group with the current device as the group owner.</td></tr><tr><td><code>removeGroup()</code></td><td>Removes the current peer-to-peer group.</td></tr><tr><td><code>requestGroupInfo()</code></td><td>Requests peer-to-peer group information.</td></tr><tr><td><code>discoverPeers()</code></td><td>Initiates peer discovery</td></tr><tr><td><code>requestPeers()</code></td><td>Requests the current list of discovered peers.</td></tr></tbody></table> <p><code>WifiP2pManager</code> methods let you pass in a listener, so that the Wi-Fi P2P framework can notify your activity of the status of a call. The available listener interfaces and the corresponding <code>WifiP2pManager</code> method calls that use the listeners are described in the following table:</p> <p>(See Android Developer Webpage, “Wi-Fi Peer-to-Peer”)^{lxv}</p>	Method	Description	<code>initialize()</code>	Registers the application with the Wi-Fi framework. This must be called before calling any other Wi-Fi P2P method.	<code>connect()</code>	Starts a peer-to-peer connection with a device with the specified configuration.	<code>cancelConnect()</code>	Cancels any ongoing peer-to-peer group negotiation.	<code>requestConnectInfo()</code>	Requests a device's connection information.	<code>createGroup()</code>	Creates a peer-to-peer group with the current device as the group owner.	<code>removeGroup()</code>	Removes the current peer-to-peer group.	<code>requestGroupInfo()</code>	Requests peer-to-peer group information.	<code>discoverPeers()</code>	Initiates peer discovery	<code>requestPeers()</code>	Requests the current list of discovered peers.
Method	Description																				
<code>initialize()</code>	Registers the application with the Wi-Fi framework. This must be called before calling any other Wi-Fi P2P method.																				
<code>connect()</code>	Starts a peer-to-peer connection with a device with the specified configuration.																				
<code>cancelConnect()</code>	Cancels any ongoing peer-to-peer group negotiation.																				
<code>requestConnectInfo()</code>	Requests a device's connection information.																				
<code>createGroup()</code>	Creates a peer-to-peer group with the current device as the group owner.																				
<code>removeGroup()</code>	Removes the current peer-to-peer group.																				
<code>requestGroupInfo()</code>	Requests peer-to-peer group information.																				
<code>discoverPeers()</code>	Initiates peer discovery																				
<code>requestPeers()</code>	Requests the current list of discovered peers.																				

Discovering peers

To discover peers that are available to connect to, call `discoverPeers()` to detect available peers that are in range. The call to this function is asynchronous and a success or failure is communicated to your application with `onSuccess()` and `onFailure()` if you created a `WifiP2pManager.ActionListener`. The `onSuccess()` method only notifies you that the discovery process succeeded and does not provide any information about the actual peers that it discovered, if any.

```
mManager.discoverPeers(channel, new WifiP2pManager.ActionListener() {
    @Override
    public void onSuccess() {
        ...
    }

    @Override
    public void onFailure(int reasonCode) {
        ...
    }
});
```

If the discovery process succeeds and detects peers, the system broadcasts the `WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION` intent, which you can listen for in a broadcast receiver to obtain a list of peers. When your application receives the `WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION` intent, you can request a list of the discovered peers with `requestPeers()`. The following code shows how to set this up:

```
PeerListListener myPeerListListener;
...
if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
    // request available peers from the wifi p2p manager. This is an
    // asynchronous call and the calling activity is notified with a
    // callback on PeerListListener.onPeersAvailable()
    if (mManager != null) {
        mManager.requestPeers(mChannel, myPeerListListener);
    }
}
```

The `requestPeers()` method is also asynchronous and can notify your activity when a list of peers is available with `onPeersAvailable()`, which is defined in the `WifiP2pManager.PeerListListener` interface. The `onPeersAvailable()` method provides you with an `WifiP2pDeviceList`, which you can iterate through to find the peer that you want to connect to.

(See Android Developer Webpage, “Wi-Fi Peer-to-Peer”)^{lxvi}

As another example, as shown below, Bluetooth APIs provide a mechanism for searching for Network Services to be used by application software on a WLAN Device.

Finding Devices

Using the `BluetoothAdapter`, you can find remote Bluetooth devices either through device discovery or by querying the list of paired (bonded) devices.

Device discovery is a scanning procedure that searches the local area for Bluetooth enabled devices and then requesting some information about each one (this is sometimes referred to as "discovering," "inquiring" or "scanning"). However, a Bluetooth device within the local area will respond to a discovery request only if it is currently enabled to be discoverable. If a device is discoverable, it will respond to the discovery request by sharing some information, such as the device name, class, and its unique MAC address. Using this information, the device performing discovery can then choose to initiate a connection to the discovered device.

(See Android Developer Webpage, "Bluetooth")^{lxvii}

- **Service discovery:**

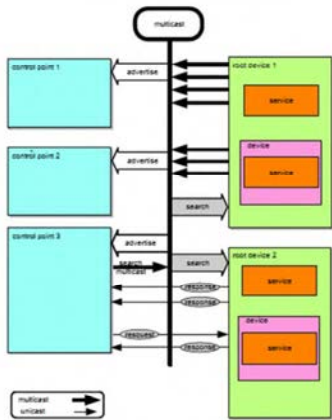
Once the local device has discovered at least one remote device, it can begin to search for available services - Bluetooth applications can use to accomplish useful tasks. Because service discovery is much like device discovery, `DiscoveryAgent` also provides methods to discover services on a Bluetooth server device, and to initiate service-discovery transactions. Note that the API provides mechanisms to search for services on remote devices, but not for services on the local device.

The `servicesDiscovered()` and `serviceSearchCompleted()` methods of `DiscoveryAgent` must be implemented. They will handle the events occurring when services are found or when the service discovery completes.

(See Samsung API Guide JSR 82- Bluetooth, Version .9 at 13-14)

As another example, as shown below, DLNA and UPnP provide a mechanism for searching for Network Services to be used by application software on a WLAN Device.

Figure 1-1: – Discovery architecture



When a device knows it is newly added to the network, it shall multicast a number of discovery messages advertising itself, its embedded devices, and its services (initial announce). Any interested control point can listen to the standard multicast address for notifications that new capabilities are available. A multi-homed device shall multicast the discovery messages on all UPnP-enabled interfaces. A multi-homed control point is allowed to listen to the standard multicast address on one, some or all of its UPnP-enabled interfaces.

When a new control point is added to the network, it is allowed to multicast a discovery message searching for interesting devices, services, or both. All devices shall listen to the standard multicast address for these messages and shall respond if any of their root devices, embedded devices or services matches the search criteria in the discovery message. In addition, a control point is allowed to unicast a discovery message to a specific IP address on port 1900 or on the port specified by the optional SEARCHPORT UPNP.ORG header field (which supersedes port 1900 for this use), searching for a UPnP device or service at that specific IP address. This action presumes the control point already knows the device at this IP address is a UPnP device (which listens on the appropriate port). The control point can use unicast search for a number of applications. A unicast search can quickly confirm a specific device and provide the corresponding discovery information (e.g. UUID, URL) of this device.

(See UPnP Device Architecture 2.0, Sept 1, 2014 at 19)

As another example, as shown below, the AllShare Framework and Media Control Framework provide a mechanism for searching for Network Services to be used by application software on a WLAN Device.

Samsung Web API: Provider

© 2013 Samsung Electronics Co., Ltd. All rights reserved.

Introduction

This API allows developers to share media contents between DLNA devices.

It provides the operations as follows:

- Browse or search contents which are provided from all available digital media servers (DMS)
- Download a content from another DMS to my local device.

(See Samsung’s AllShare Web API, “Provider”)^{lxviii}

4.4. Browsing and Sharing Media Contents from a Media Control Framework Device

To get a content list from a media content provider:

1. Call the browse and search asynchronous methods of the SmcProvider class.
 - The browse method returns a list of content items according to the directory structure in the Media Control Framework device.
 - The search method returns contents matching the specified search conditions.

(See Samsung Media Control Programming Guide Version 1.0.1 at 18-20)

This section defines AllShare APIs that would be used to develop convergence services such as media sharing and control sharing. The AllShare APIs provides the below features applicable to any type of devices including mobile, smart tv, pc, etc:

- Media Sharing
 - discover DLNA devices in a network.
 - share media contents (picture, audio, video, etc) to other DLNA devices in a network.
 - browse and search the media contents shared by DLNA devices in a network.
 - playback media contents on a DLNA device in a network.
- Control Sharing
 - discover Smart TV in a network.
 - control TV and TV web browser remotely like a remote controller, mouse and keyboard did.
 - control TV viewer in detail as an extension of ImageViewer (DMR).

(See Samsung’s AllShare Web API, “Provider”)^{lxix}

Developers can discover DLNA devices through APIs provided by the DeviceFinder module. A DLNA device can be a digital media server (DMS) or a digital media render (DMR). DMS devices, represented by the Provider module, share media content located on the devices to other DLNA devices. DMR devices, represented by the AVPlayer module or ImageViewer module, are able to play the media content shared by DMS devices. Detailed information on DLNA is available on www.dlna.org.

Developers can browse and search shared media content through APIs provided by the Provider module. The media content can be image files or audio (video) files located on the DMS device that shares the files. The media content is represented by the Item module. In order to playback the files, a proper type of media player is required as the media type. Developers must use the APIs provided by the ImageViewer module for the image type and the APIs provided by the AVPlayer module for the audio or video type.

Media Sharing APIs

Feature	Subfeature	Module	Description
Media Sharing	Content Discovery	Provider	This interface provides interfaces to browse and search media content shared by DLNA devices and to download content from a remote DMS.
		Item	This API provides interfaces to retrieve information about media content.
	Playback Content	AVPlayer	This interface provides interfaces to play audio (or video) content on a DLNA device.
		ImageViewer	This interface provides interfaces to play image content on a DLNA device.

(See Samsung’s AllShare Web API, “Introduction”)^{bxx}

2.2. Browsing and sharing media contents from a AllShare device

To get a content list from a media content provider, you can use `browse()` and `search()` asynchronous methods of the `Provider` class. `browse()` method provides a list of content items according to directory structure in the AllShare device. `search()` method allows to search content with specified conditions.

Below is sample code showing the use of `browse()` method.

```

Sample Code
DeviceFinder deviceFinder = serviceProvider.getDeviceFinder();
ArrayList<Device> providerList = deviceFinder.getDevices(DeviceType.DEVICE_PROVIDER);
Provider selectedProvider = (Provider)providerList.get(0);

/** enter root folder as input for the first browse on a provider */
selectedProvider.browse(selectedProvider.getRootFolder(), 0, 10);
    
```

`browse()` requires `parentFolderItem`, `startIndex` and `requestCount` as input parameters. AllShare Framework returns the requested number of items from the content list via `IProviderBrowseResponseListener`. When you determine `requestCount`, you should be careful because it affects browse response speed.

You can get browse response by registering a listener in the following way:

```

Sample Code
IProviderBrowseResponseListener mBrowseResponseListener = new IProviderBrowseResponseListener()
{
    @Override
    public void onBrowseResponseReceived(ArrayList<Item> items, int requestedStartIndex,
        int requestCount, Item requestedFolderItem,
        boolean endOfItems, ERROR err)
    {
    }
};
selectedProvider.setBrowseItemsResponseListener(mBrowseResponseListener);
    
```

Possible values of the *MediaType* Item are listed below.

Item Type Enum	Description
ITEM_FOLDER	Folder item, that may contain any other Item
ITEM_AUDIO	Representation of the Item is an audio file
ITEM_IMAGE	Representation of the Item is an image file
ITEM_VIDEO	Representation of the Item is a video file
ITEM_UNKNOWN	Unknown item

search() can be used in the following way:

```

Sample Code
DeviceFinder deviceFinder = serviceProvider.getDeviceFinder();
ArrayList<Device> providerList = deviceFinder.getDevices(DeviceType.DEVICE_PROVIDER);
Provider selectedProvider = (Provider)providerList.get(-);

SearchCriteria.Builder builder = new SearchCriteria.Builder();
builder.addItemType(MediaType.ITEM_AUDIO);
builder.setKeyword("love");
SearchCriteria criteria = builder.build();

selectedProvider.search(SearchCriteria searchString, int startIndex, int requestCount);

```

search() requires *searchCriteria*, *startIndex* and *requestCount* as input parameters. AllShare Framework returns the requested number items from the content list via *iProviderSearchResponseListener*. The list starts with the index specified by the *searchCriteria* condition.

(See AllShare Framework: Developer’s Guide Version 2.0 at 13-14)

As another example, as shown below, Samsung’s Chord Framework provides a mechanism for searching for Network Services to be used by application software on a WLAN Device.

4.1.1. Discovering Chord Peers

Each Chord node transmits a UDP broadcast periodically, and parses broadcast messages from other nodes to discover all the nodes on the same subnet.

Devices running Chord-based applications join the public channel automatically.

A node cannot receive a UDP broadcast if it is in LCD-off status. Set the node status to LCD-on to enable the node to discover other nodes while the application is running. To do this, use the normal methods enabled by `android.os.PowerManager.WakeLock`.

(See Chord Programming Guide, Version 2.0.1 at 14)

As another example, as shown below, Samsung’s Accessory Framework provides a mechanism for searching for Network Services to be used by application software on a WLAN Device.

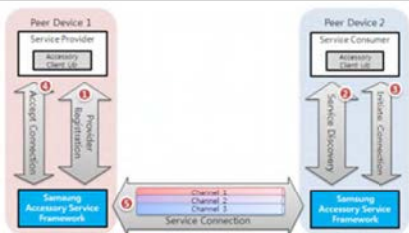


Figure 3. Functional flow between Service Provider and Service Consumer

The Service Provider and Service Consumer applications register their service capabilities with the Samsung Accessory Service Framework. The Samsung Accessory Service Framework advertises and exchanges the capabilities of the registered Service Providers and Service Consumers.

The Service Consumer looks for Service Providers of interest, and queries the Samsung Accessory Service Framework, which in turn queries the services offered by connected Accessory Devices.


The Service Consumer attempts to establish a Service Connection with the Service Provider. A Service Provider can also try to establish Service Connections with Service Consumers.

The Service Provider decides to accept or reject the Service Connection request. If the Service Provider attempted to establish a connection, the Service Consumer decides to accept or reject the Service Connection request.

The Service Connection is established, creating all the Service Channels defined by the associated Accessory Service Profile. The Service Consumer and Service Provider use the established Service Connection to read and write data following the associated Accessory Service Profile specification on the Service Channels.

(See Accessory Programming Guide, Version 2.1.11 at 6)

	<p style="text-align: center;">4.3. Finding a Matching Accessory Peer Agent and Initiating a Service Connection</p> <p>Your Service Provider or Service Consumer application can search for matching Accessory Peer Agents by calling the SAAgent.FindPeerAgents() method. Matching Accessory Peer Agents have the same Accessory Service Profile, i.e., Notification Service or Weather Service, and have a complementary provider or consumer relationship with the calling Accessory Peer Agent. Accessory Peer Agents with different Accessory Service Profiles for Service Providers or Service Consumers do not "match" and cannot be connected with each other. If two Accessory Peer Agents have the same Accessory Service Profile with different versions, however, they are still considered to "match". For example, Notification Service Consumer that implements the Notification Service Profile version 2.0 and a Notification Service Provider that implements the Notification Service Profile version 1.0, "match".</p> <p>If a matching Accessory Peer Agent is found, the calling Accessory Peer Agent is notified with the onFindPeerAgentResponse() callback method. If multiple matching Accessory Peer Agents are found, the callback occurs multiple times, one for each matching Accessory Peer Agent. If no Accessory Peer Agent is found, the calling Accessory Peer Agent is notified with the same callback method, but the peerAgent parameter is null and the result parameter includes the reason why there is no match found.</p> <p style="text-align: right;">(See Accessory Programming Guide, Version 2.1.11 at 25)</p>
Claim 26	
<p>The system of claim 25, wherein the first wireless device provides execution space for executable software from the second wireless device.</p>	<p>The content corresponding to Claim 25 is hereby incorporated by reference. A WLAN Device is the claimed first wireless device. WLAN Devices include memory that provides execution space for executable software provided from Samsung Relevant Devices. This memory may be, for example, random-access memory (RAM), dynamic random-access memory (DRAM), nonvolatile random-access memory (NVRAM), synchronous dynamic random-access memory (SDRAM), double data rate synchronous dynamic random access memory (DDR SDRAM), MLC NAND flash memory, SLC NAND flash memory, NAND flash memory, NOR flash memory, flash memory, read-only memory (ROM), programmable read-only memory (PROM), Mask ROM (MROM), and/or electronically erasable programmable read-only memory (EEPROM).</p> <p>For example, as shown below, the Samsung Galaxy Tab includes SanDisk SDIN4C2-16G NAND Flash memory and 1 Gb of OneDram and 3 Gb of mobile DDR, which includes execution space.</p>

	 <p style="text-align: right;">(See iFixit, “Galaxy Tab Teardown”)^{xxi}</p>
Claim 27	
<p>The system of claim 25, wherein the security software component is a firewall software component to control access to the cellular network.</p>	<p>The content corresponding to Claim 25 is hereby incorporated by reference. Samsung Relevant Devices include one or more firewall software components to control access to the cellular. For example, the Samsung LTE Mobile HotSpot Pro has a firewall software component as shown below.</p> <p style="text-align: center;">Port Forwarding Panel</p> <p>Port Forwarding allows incoming traffic (from the Internet) to be forwarded to a particular PC or device on your local WLAN. Normally, incoming traffic from the Internet is blocked by the Firewall.</p> <p>You need to use Port Forwarding to allow Internet users to access running services such as a Web server, FTP server, E-mail server, etc...</p> <p>For some online applications (such as games), Port Forwarding must be used in order for the game to function correctly.</p> <p style="text-align: right;">(See Samsung LTE Mobile Hotspot PRO User Manual at 35)</p>
Claim 28	
<p>The system of claim 25, wherein the security software component is a virtual private network (“VPN”) to control access to the cellular network.</p>	<p>The content corresponding to Claim 25 is hereby incorporated by reference. Samsung Relevant Devices include one or more VPN software components. For example, the Samsung Galaxy S4 includes a VPN software component.</p> <p style="text-align: center;">VPN</p> <p>Set up and connect to virtual private networks (VPNs).</p> <p style="text-align: right;">(See Samsung Galaxy S4 User Manual at 121)</p>

Claim 30	
<p>The system of claim 25, wherein the first short-range radio signal is selected from a group consisting of a HomeRF signal, an 802.11 signal and Bluetooth™.</p>	<p>The content corresponding to Claim 25 is hereby incorporated by reference. As described above, the first short-range radio signal is selected from a group consisting of a HomeRF signal, an 802.11 signal, and Bluetooth. For example, as described above, WLAN Devices communicate with Samsung Relevant Devices utilizing Wireless Hotspot Features via Wi-Fi 802.11 and/or Bluetooth.</p>
Claim 34	
<p>A handheld device for providing a short distance wireless network, comprising:</p>	<p>Although the preamble to Claim 34 does not limit the scope of the claim, Samsung provides Samsung Relevant Devices that are handheld devices for providing a short distance wireless network.</p> <p>The Wireless Hotspot Features of Samsung Relevant Devices provides WLAN Devices with a short distance wireless network (e.g., 802.11 and/or Bluetooth networks). WLAN Devices connect to Samsung Relevant Devices via short distance radio waves using the 802.11 and/or Bluetooth protocols and use the Samsung Relevant Device as a wireless access point to cellular networks.</p> <p>For example, the Samsung Galaxy S4 includes “Portable Wi-Fi hotspot” feature, which allows WLAN Devices to connect to the Samsung Galaxy S4 via Wi-Fi and use its cellular connection. (See Samsung Galaxy S4 User Manual at 121)</p> <p>Similarly, the Samsung ATIV S Neo includes an “Internet Sharing” feature, which allows WLAN Devices to connect to the Samsung ATIV S Neo via Wi-Fi and use its cellular connection. (See Samsung ATIV S Neo User Manual (Sprint) at 61)</p> <p>Samsung Relevant Devices also provides short distance networks to WLAN Devices via Wi-Fi Direct (P2P), Bluetooth, and/or reasonably similar protocols.</p>
<p>a storage device;</p>	<p>Samsung Relevant Devices include memory, which is the claimed storage device. This memory may be, for example, random-access memory (RAM), dynamic random-access memory (DRAM), nonvolatile random-access memory (NVRAM), synchronous dynamic random-access memory (SDRAM), double data rate synchronous dynamic random access memory (DDR SDRAM), MLC NAND flash memory, SLC NAND flash memory, NAND flash memory, NOR flash memory, flash memory, read-only memory (ROM), programmable read-only memory (PROM), Mask ROM (MROM), and/or electronically erasable programmable read-only memory (EEPROM).</p>

As shown below, the Samsung Galaxy S4, for example, includes Samsung K3QF2F200C-XGCE DDR3 DRAM (LPDDR3), Samsung KMV3W000LM-B310 Multichip memory – 64 MB Mobile DDR SDRAM, 16 GB MLC NAND Flash, and Atmel UC128L5-U 32 bit Microcontroller with 128 kb Flash memory.



(See Techinsights Webpage, “Samsung Galaxy S4 Teardown”)^{lxxii}

a processor, coupled to the storage device; and, the storage device to store a software component; and, the processor operative with the software component to:

Samsung Relevant Devices each include one or more processors, such as application and baseband processors. For example, the Samsung Galaxy S4 includes the Exynos 5410 Eight-Core Processor, the Intel PMB9820 baseband processor, and the Broadcom BCM4335 Wi-Fi 802.11, dual-band, DLNA, Wi-Fi Direct, Wi-Fi Hot Spot all-in-one processor as shown below. These processors, as shown below, are coupled to the memory that is describe above.



(See Techinsights Webpage, “Samsung Galaxy S4 Teardown”)^{lxxiii}

	<p>These processors operate with the operating system software (e.g., Android or Windows), Wireless Hotspot Feature software, and/or other Network Service application software that are stored on the memory of the Samsung Relevant Devices to perform the claim elements described below.</p>
<p>provide an Internet Protocol (“IP”) data packet from the handheld device to a terminal using short-range radio signals,</p>	<p>WLAN Devices are the claimed terminals. When WLAN Devices use the Wireless Hotspot Features of Samsung Relevant Devices to access the Internet, the processors of Samsung Relevant Devices together with operating system software (e.g., Android or Windows), Wireless Hotspot Feature software, and/or other Network Service software provide IP data packets to one or more WLAN Devices via short range RF signals transmitted by Wi-Fi and/or Bluetooth transceivers pursuant to IEEE, Bluetooth, IPv4, and/or IPv6 standards.</p> <p>For example, in the Samsung Galaxy S4, the Broadcom BCM4335 Wi-Fi 802.11, dual-band, DLNA, Wi-Fi Direct, Wi-Fi Hot Spot all-in-one processor processes IP data packets onto Wi-Fi short-range RF signals.</p> <div data-bbox="625 951 1169 1276" data-label="Image"> </div> <p>(See Techinsights Webpage, “Samsung Galaxy S4 Teardown”)^{lxiv}</p> <p>In addition, other Network Services provided to WLAN Devices from Samsung Relevant Devices provide IP data packets to WLAN Devices, such as account authentication, user preferences, cloud-based data (e.g., email, calendar, messages, media, etc.), and other information when they are connected to Samsung Relevant Devices via Wireless Hotspot Features, Wi-Fi Direct (P2P), Bluetooth, or reasonably similar protocols.</p>
<p>control access between the short distance</p>	<p>The processors together with operating system software (e.g., Android or Windows) and/or Wireless Hotspot Feature software control access between the short distance wireless network and a cellular network. WLAN</p>

<p>wireless network and a cellular network,</p>	<p>Devices operate on the short distance wireless network created by the Samsung Relevant Device. WLAN Devices can access the cellular network via the Samsung Relevant Devices only if the communication is authorized by the Samsung Relevant Device.</p> <p>For example, Samsung Relevant Devices include software that utilizes MAC filtering, port filtering, VPNs (Virtual Private Networks), Bluetooth pairing, and/or adheres to security protocols such as Wi-Fi Protected Access (WPA) and/or WPA II. Samsung Relevant Devices also include Network Service passwords, such as a Samsung Link password, which control access between the cellular network and the Samsung Relevant Device.</p>
<p>translate between a first IP address provided to the handheld device and a second IP address for the terminal provided by the handheld device in the short distance wireless network,</p>	<p>The processors together with operating system software (e.g., Android or Windows) and/or Wireless Hotspot Feature software translate between a first IP addresses provided from the cellular network and a second IP address for the WLAN Device provided over a Wi-Fi and/or Bluetooth network.</p> <p>This portion of the claim is further discussed in relation to the “network address translator software component” of Claim 1, which discussion is hereby incorporated by reference.</p>
<p>enumerate a list of services available from the handheld device and the terminal, wherein the handheld device and terminal register services available on the list, and</p>	<p>The processors together with operating system software (e.g., Android or Windows), Wireless Hotspot Feature software, and/or other Network Service application software enumerate a list of Network Services available from and registered by a Samsung Relevant Device and one or more WLAN Devices.</p> <p>As discussed for the “service repository software component” of Claim 1, which discussion is hereby incorporated by reference, Samsung Relevant Devices include service repository software components that identify (enumerate) Network Services available from WLAN Devices and Samsung Relevant Devices. These Network Services are registered by the WLAN Devices and Samsung Relevant Devices pursuant to the various APIs, Frameworks, and/or protocols discussed in connection with Claim 14, which discussion is hereby incorporated by reference.</p>
<p>search the list of services for a service to be used by an application software component stored on the terminal.</p>	<p>The processors together with operating system software, Wireless Hotspot Feature software, and/or Network Service application software search the list of services to be used by a software application component stored on one or more WLAN Device.</p> <p>As discussed in connection with the phrase “wherein the service repository software searches for a service” for Claim 25, which discussion is hereby incorporated by reference, the processors together with operating system software (e.g., Android or Windows), Wireless Hotspot Feature software, and/or Network Service software search the list of services to be used by a Network Service application component stored on one or more WLAN Device.</p>

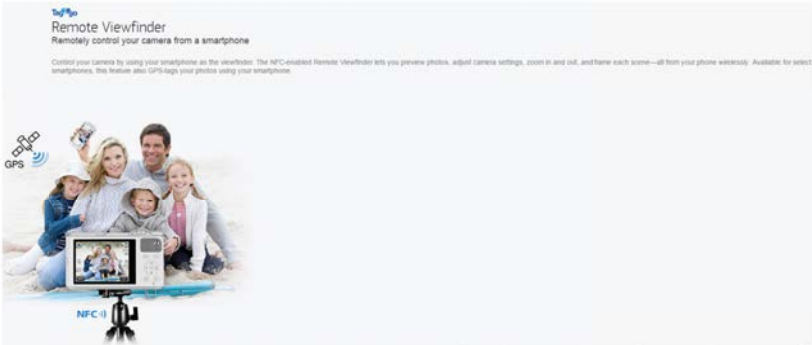
Claim 39	
The device of claim 34, wherein the search includes searching the list of services by class, attribute or instance.	The content corresponding to Claim 34 is hereby incorporated by reference. As discussed in connection with Claim 22, which discussion is hereby incorporated by reference, Samsung Relevant Devices search the list of services by class, attribute and instance of the service.
Claim 40	
The device of claim 34, wherein the software component includes a plug and play software component to identify the terminal in the short distance wireless network and obtain the application software component for the terminal.	The content corresponding to Claim 34 is hereby incorporated by reference. Samsung Relevant Devices include software components with plug and play software components that identify WLAN Devices in short distance wireless networks and obtain Network Service application software for the WLAN Devices. For example, Samsung Relevant Devices include UPnP and/or DLNA APIs or reasonably similar APIs that may be used to load and execute software for WLAN Devices that are compatible with DLNA and/or UPnP. In addition, Samsung Relevant Devices and WLAN Devices include AllShare/Link, Samsung Print Services, and/or Samsung PC Share Manager software components that are DLNA or UPnP certified and offer plug and play functionality. As another example, Samsung Relevant Devices include software such as Wireless Hotspot Feature software that identifies WLAN Devices and provides DHCP software components to the identified WLAN Devices.
Claim 41	
The device of claim 34, wherein the software component includes a PIN number management software component to provide a PIN number used in pairing the handheld device to the terminal in the short distance wireless network.	<p>The content corresponding to Claim 34 is hereby incorporated by reference.</p> <p>The software components include PIN number management software to provide PIN numbers used in pairing the Samsung Relevant Device and the WLAN Device. For example, Bluetooth Tethering requires a pairing PIN Code.</p> <p>Pairing with other Bluetooth devices</p> <p>On the Applications screen, tap Settings → Connections → Bluetooth → Scan, and detected devices are listed. Select the device you want to pair with, and then accept the auto-generated passkey on both devices to confirm.</p> <p style="text-align: right;">(See Samsung Galaxy S4 User Manual at 68)</p>
Claim 42	
A first wireless handheld device, comprising:	Although the preamble to Claim 42 does not limit the scope of the claim, Samsung provides Samsung Relevant Devices that are first handheld devices.

a storage device;	As described for the “storage device” of Claim 34, which description is hereby incorporated by reference, Samsung Relevant Devices include memory, which is the claimed storage device.
a processor, coupled to the storage device; and, the storage device to store a software component; and, the processor operative with the software component to:	As described for the “processor, coupled to the storage device” of Claim 34, which description is hereby incorporated by reference, Samsung Relevant Devices each include one or more processors that are coupled to the memory. These processors operate with the operating system software (e.g., Android or Windows), Wireless Hotspot Feature software, and/or other Network Service software that are stored on the memory of the Samsung Relevant Devices to perform the claim elements described below.
access the Internet through a cellular network,	<p>The processors together with operating system software (e.g., Android or Windows) and/or Wireless Hotspot Feature software provide WLAN Devices with access to Internet Devices via wireless local area radio signals (e.g., 802.11 and/or Bluetooth) and cellular networks.</p> <p>As shown below, the Samsung Galaxy S4 can use mobile data to access the internet via cellular networks.</p> <p style="text-align: center;">Mobile networks</p> <ul style="list-style-type: none"> • Mobile data: Use to allow packet switching data networks for network services. <p style="text-align: right;">(See Samsung Galaxy S4 User Manual at 121)</p> <p>In addition, the Samsung Galaxy S4 includes “Portable Wi-Fi hotspot” feature, which allows WLAN Devices to connect to the Samsung Galaxy S4 via Wi-Fi and use its cellular connection to access the Internet. (See Samsung Galaxy S4 User Manual at 121)</p> <p>Similarly, the Samsung ATIV S Neo includes an “Internet Sharing” feature, which allows WLAN Devices to connect to the Samsung ATIV Neo via Wi-Fi and use its cellular connection to access the Internet. (See Samsung ATIV S Neo User Manual (Sprint) at 61)</p>
provide a first short-range radio signal to a second wireless handheld device and a second short-range radio signal to a third wireless	The processors together with operating system software (e.g., Android or Windows) and/or Wireless Hotspot Feature software of the Samsung Relevant Devices provide a first short-range radio frequency signal pursuant to 802.11 and/or Bluetooth standards to a second wireless handheld device (a first WLAN Device) and a second short-range radio frequency signal pursuant to 802.11 and/or Bluetooth standards to a third wireless handheld device (a second WLAN Device). Android’s Wireless Hotspot Feature, for example, allows up to 10 WLAN Devices to connect at one time to a single Samsung Relevant Device.

handheld device,	In addition, Samsung Relevant Devices provide a first and second short-range radio signals to a first and second WLAN Device when providing or receiving Network Services from the WLAN Devices via the Bluetooth PAN Profile, Wi-Fi Direct (P2P), or reasonably similar protocol.
control access between the Internet and the first, second and third wireless handheld devices,	<p>The processors together with operating system software (e.g., Android or Windows) and/or Wireless Hotspot Feature software control access between the Internet Devices and the first and second WLAN Devices. WLAN Devices can access the Internet via the Samsung Relevant Devices only if the communication is authorized by the Samsung Relevant Device.</p> <p>For example, Samsung Relevant Devices include software that utilizes MAC filtering, port filtering, VPNs (Virtual Private Networks), Bluetooth pairing, and/or adheres to security protocols such as Wi-Fi Protected Access (WPA) and/or WPA II. Samsung Relevant Devices also include Network Service passwords, such as a Samsung Link password, which control access between the cellular network and the Samsung Relevant Device.</p>
translate between a first Internet Protocol (“IP”) address provided to the first wireless handheld device from the cellular network and a second address for the second wireless handheld device provided by the first wireless handheld device, and a third address for the third wireless handheld device provided by the first wireless device,	<p>The processors together with operating system software (e.g., Android or Windows) and/or Wireless Hotspot Feature software translate a first IP address received from an Internet Device and/or the cellular network to a second IP address that is then provided to the first WLAN Device and to a third IP address that is then provided to a second WLAN Device.</p> <p>This portion of the claim is further discussed in connection with the “network address translator software component” of Claim 1, which discussion is hereby incorporated by reference.</p>
enumerate a list of services available from the first, second and third wireless handheld devices, wherein the first,	<p>The processors together with operating system software (e.g., Android or Windows), Wireless Hotspot Feature software, and/or application software enumerate a list of services available from and registered by a Samsung Relevant Device and the first and second WLAN Devices.</p> <p>As discussed for the “service repository software component” of Claim 1, which discussion is hereby incorporated</p>

<p>second and third wireless handheld devices register services available on the list, and</p>	<p>by reference, Samsung Relevant Devices include service repository software components that identify (enumerate) Network Services available from WLAN Devices and Samsung Relevant Devices. These Network Services are registered by the WLAN Devices and Samsung Relevant Devices pursuant to the various APIs, Frameworks, and/or protocols discussed in connection with Claim 14, which discussion is hereby incorporated by reference.</p>
<p>search the list of services for a class of service to be used by an application software component at a particular time, the application software component stored on the second wireless handheld device.</p>	<p>As discussed in connection with the phrase “wherein the service repository software searches for a service” of Claim 25, which discussion is hereby incorporated by reference, the processors together with operating system software (e.g., Android or Windows), Wireless Hotspot Feature software, and/or Network Service software search the list of services for a class of service to be used by an application stored on the first WLAN Device at a particular time.</p>
<p>Claim 43</p>	
<p>The first wireless handheld device of claim 42, wherein the first wireless handheld device includes a service logical driver corresponding to a service available from the third wireless device, and the application software component uses the service logical driver to obtain the service from the third wireless device.</p>	<p>The content corresponding to Claim 42 is hereby incorporated by reference. A Samsung Relevant Device is the claimed first wireless handheld device. Samsung Relevant Devices include service logical drivers that correspond to services available from a second WLAN Device (a third wireless handheld device) and the application software component of the Samsung Relevant Devices uses the logical drivers to obtain services from the second WLAN Device.</p> <p>For example, as discussed in connection with the “service repository software component” of Claim 1, which discussion is hereby incorporated by reference, a Samsung Relevant Device can obtain Network Services from a second WLAN Device as described in various protocols and APIs. Also, as discussed in connection with Claim 15, which discussion is also incorporated by reference, Network Services applications may also use connect-to-service or connect-by-name APIs to use a Network Service provided by the second WLAN Device.</p>
<p>Claim 46</p>	
<p>The first wireless handheld device of claim 42, wherein the second wireless handheld device is a thin terminal.</p>	<p>The content corresponding to Claim 42 is hereby incorporated by reference.</p> <p>A WLAN Device is the claimed second wireless device. A WLAN Device may be a thin terminal such as a printer or camera. For example, Samsung Relevant Devices can communicate wirelessly with printers via applications such as the Samsung Print Service Plugin. (See Samsung Galaxy S4 User Manual at 99)</p>

Samsung Relevant Devices can also communicate wirelessly with smart cameras via applications such as remote viewfinder.



(See Samsung’s Webpage, “Remote Viewfinder”)^{lxv}



(See Samsung Smart Camera YouTube Advertisement)^{lxvi}

-
- ⁱ <http://www.wired.com/2010/05/android-22-froyo-features-usb-wi-fi-tethering/> (last visited Jan. 7, 2015).
 - ⁱⁱ <https://www.windowsphone.com/en-us/how-to/wp7/start/share-my-connection> (last visited Jan. 7, 2015).
 - ⁱⁱⁱ <http://developer.android.com/training/connect-devices-wirelessly/nsd.html> (last visited Dec. 5, 2014).
 - ^{iv} <http://developer.android.com/about/versions/android-4.1.html> (last visited Jan. 7, 2015).
 - ^v <http://developer.android.com/training/connect-devices-wirelessly/nsd.html> (last visited Dec. 5, 2014).
 - ^{vi} <http://developer.android.com/training/connect-devices-wirelessly/nsd.html#discover> (last visited Dec. 5, 2014).
 - ^{vii} <https://developer.android.com/guide/topics/media/mediarouteprovider.html> (last visited Dec. 30, 2014).
 - ^{viii} <http://developer.android.com/reference/android/media/MediaRouter.html> (last visited Dec. 30, 2014).
 - ^{ix} <https://developer.android.com/guide/topics/media/mediarouteprovider.html> (last visited Dec. 30, 2014).
 - ^x <https://developer.android.com/guide/topics/media/mediarouteprovider.html> (last visited Dec. 30, 2014).
 - ^{xi} <https://developer.android.com/reference/android/support/v7/media/MediaRouteProvider.html> (last visited Dec. 30, 2014).
 - ^{xii} <http://developer.android.com/training/connect-devices-wirelessly/nsd-wifi-direct.html> (last visited Dec. 5, 2014).
 - ^{xiii} <http://developer.android.com/about/versions/android-4.0-highlights.html> (last visited Jan. 7, 2015).
 - ^{xiv} <http://developer.android.com/training/connect-devices-wirelessly/nsd-wifi-direct.html> (last visited Dec. 5, 2014).
 - ^{xv} <http://developer.android.com/guide/topics/connectivity/bluetooth.html> (last visited Dec. 5, 2014).
 - ^{xvi} <http://developer.android.com/reference/android/bluetooth/BluetoothClass.html> (last visited Dec. 5, 2014).
 - ^{xvii} <http://msdn.microsoft.com/en-us/library/windows/apps/jj207060%28v=vs.105%29.aspx> (last visited Dec. 5, 2014).
 - ^{xviii} <http://msdn.microsoft.com/en-us/library/windows/apps/windows.networking.proximity.peerfinder.aspx> (last visited Dec. 5, 2014).
 - ^{xix} <http://msdn.microsoft.com/en-us/library/windows/apps/windows.networking.proximity.peerwatcher.aspx> (last visited Dec. 5, 2014).
 - ^{xx} <http://msdn.microsoft.com/en-us/library/windows/apps/windows.devices.wifidirect.wifidirectdevice.aspx> (last visited Dec. 5, 2014).
 - ^{xxi} [http://msdn.microsoft.com/en-us/library/windows/apps/jj207007\(v=vs.105\).aspx#BKMK_BluetoothAPI](http://msdn.microsoft.com/en-us/library/windows/apps/jj207007(v=vs.105).aspx#BKMK_BluetoothAPI) (last visited Dec. 5, 2014).
 - ^{xxii} <http://developer.samsung.com/galaxy#media-control> (last visited Jan. 7, 2015).
 - ^{xxiii} <http://developer.samsung.com/resources/media-control> (last visited Jan. 7, 2015).
 - ^{xxiv} http://img-developer.samsung.com/onlinedocs/samsung_webapi_guide_public_2.0/html/index.html (last visited Dec. 5, 2014).
 - ^{xxv} http://img-developer.samsung.com/onlinedocs/samsung_webapi_guide_public_2.0/html/index.html (last visited Dec. 5, 2014).
 - ^{xxvi} http://img-developer.samsung.com/onlinedocs/samsung_webapi_guide_public_2.0/html/index.html (last visited Dec. 5, 2014).

-
- xxvii <http://developer.samsung.com/web-api> (last visited Dec. 5, 2014).
 - xxviii <http://developer.samsung.com/chord> (last visited Dec. 5, 2014).
 - xxix <http://developer.samsung.com/chord> (last visited Dec. 5, 2014).
 - xxx <http://developer.samsung.com/galaxy#accessory> (last visited Dec. 5, 2014).
 - xxxi <http://developer.samsung.com/galaxy#accessory> (last visited Dec. 5, 2014).
 - xxxii http://img-developer.samsung.com/online-docs/samsung_webapi_guide_public_2.0/html/index.html (Dec. 5, 2014).
 - xxxiii <http://developer.samsung.com/ble> (last visited Dec. 9, 2014).
 - xxxiv <http://www.dlna.org/dlna-for-industry/guidelines> (last visited Jan. 9, 2015).
 - xxxv <http://developer.android.com/reference/android/media/MediaRouter.html> (last visited Dec. 30, 2014).
 - xxxvi <http://developer.android.com/reference/android/media/MediaRouter.html> (last visited Dec. 30, 2014).
 - xxxvii <http://link.samsung.com/> (last visited Dec. 5, 2014).
 - xxxviii <http://link.samsung.com/> (last visited Dec. 5, 2014).
 - xxxix <http://content.samsung.com/us/contents/aboutn/samsungLinkIntro.do> (Dec. 5, 2014).
 - xl <http://content.samsung.com/us/contents/aboutn/samsungLinkIntro.do> (Dec. 5, 2014).
 - xli <http://www.samsung.com/us/2012-allshare-play/> (last visited Dec. 5, 2014).
 - xlii <http://content.samsung.com/us/contents/aboutn/groupPlay.do> (last visited Dec. 8, 2014).
 - xliii <http://developer.android.com/reference/java/net/InetAddress.html> (last visited Dec. 5, 2014).
 - xliv http://www.samsung.com/global/microsite/SMARTCAMERA/mobile/remote_viewfinder.html (last visited Dec. 5, 2014).
 - xlv <https://www.youtube.com/watch?v=3Jm-UE2WuEM> (last visited Nov. 10, 2014).
 - xlvi <http://developer.android.com/training/connect-devices-wirelessly/nsd.html#register> (last visited Dec. 8, 2014).
 - xlvii <https://developer.android.com/guide/topics/media/mediarouteprovider.html> (last visited Dec. 30, 2014).
 - xlviii <https://developer.android.com/reference/android/support/v7/media/MediaRouteProvider.html> (last visited Dec. 30, 2014).
 - lix <http://developer.android.com/training/connect-devices-wirelessly/nsd-wifi-direct.html> (Dec. 8, 2014).
 - ¹ <http://developer.android.com/training/connect-devices-wirelessly/nsd-wifi-direct.html> (Dec. 8, 2014).
 - ² <http://developer.android.com/reference/android/bluetooth/BluetoothClass.html> (last visited Dec. 5, 2014).
 - ³ <http://developer.android.com/reference/android/bluetooth/BluetoothClass.html> (last visited Dec. 5, 2014).
 - ⁴ <http://link.samsung.com/> (last visited Dec. 8, 2014).
 - ⁵ <http://developer.samsung.com/technical-doc/view.do?v=T000000138L> (last visited Jan. 15, 2015).
 - ⁶ <http://developer.android.com/training/connect-devices-wirelessly/nsd.html#discover> (last visited Dec. 8, 2014).
 - ⁷ <http://developer.android.com/training/connect-devices-wirelessly/nsd.html#discover> (last visited Dec. 8, 2014).
 - ⁸ <http://developer.android.com/guide/topics/connectivity/bluetooth.html> (last visited Dec. 5, 2014).
 - ⁹ <http://developer.android.com/reference/android/bluetooth/BluetoothClass.html> (last visited Dec. 5, 2014).
 - ¹⁰ <http://www.wired.com/2010/05/android-22-froyo-features-usb-wi-fi-tethering/> (last visited Jan. 7, 2015).
 - ¹¹ <https://www.windowsphone.com/en-us/how-to/wp7/start/share-my-connection> (last visited Jan. 7, 2015).
 - ¹² <http://developer.android.com/training/connect-devices-wirelessly/nsd.html#register> (last visited Dec. 8, 2014).
 - ¹³ <http://developer.android.com/training/connect-devices-wirelessly/nsd.html#register> (last visited Dec. 8, 2014).
 - ¹⁴ <https://developer.android.com/guide/topics/media/mediarouteprovider.html> (last visited Dec. 30, 2014).
 - ¹⁵ <https://developer.android.com/reference/android/support/v7/media/MediaRouteProvider.html> (last visited Dec. 30, 2014).
 - ¹⁶ <http://developer.android.com/guide/topics/connectivity/wifip2p.html> (last visited Dec. 8, 2014).

-
- ^{lxvi} <http://developer.android.com/guide/topics/connectivity/wifi2p.html> (last visited Dec. 8, 2014).
 - ^{lxvii} <http://developer.android.com/guide/topics/connectivity/bluetooth.html> (last visited Dec. 8, 2014).
 - ^{lxviii} http://img-developer.samsung.com/onlinedocs/samsung_webapi_guide_public_2.0/html/index.html (last visited Dec. 9, 2014).
 - ^{lxix} http://img-developer.samsung.com/onlinedocs/samsung_webapi_guide_public_2.0/html/index.html (last visited Dec. 9, 2014).
 - ^{lxx} http://img-developer.samsung.com/onlinedocs/samsung_webapi_guide_public_2.0/html/index.html (last visited Dec. 9, 2014).
 - ^{lxxi} <https://www.ifixit.com/Teardown/Samsung+Galaxy+Tab+Teardown/4103> (last visited Dec. 8, 2014).
 - ^{lxxii} <http://www.techinsights.com/inside-samsung-galaxy-s4/> (last visited Dec. 8, 2014).
 - ^{lxxiii} <http://www.techinsights.com/inside-samsung-galaxy-s4/> (last visited Dec. 8, 2014).
 - ^{lxxiv} <http://www.techinsights.com/inside-samsung-galaxy-s4/> (last visited Dec. 8, 2014).
 - ^{lxxv} http://www.samsung.com/global/microsite/SMARTCAMERA/mobile/remote_viewfinder.html (last visited Dec. 8, 2014).
 - ^{lxxvi} <https://www.youtube.com/watch?v=3Jm-UE2WuEM> (last visited Nov. 10, 2014).