API Specification

About This Specification

This guide specifies Cisco Wireless Phone Application Programming Interfaces (APIs) which expose wireless phone platform capabilities not available through standard Android Open Source Project (AOSP) application APIs, such as access to scanned barcode data and so on.

Note This guide is only for reference on the capabilities of Phone WebAccess APIs. See https://developer.cisco.com for more information on Software Development Kit (SDK) such as scripts and sample applications for your reference before developing the web application.

The specification is for native Android application (app) developers and assumes Android application programming competency.

As more API platform capabilities become available or as existing APIs are revised, the API version and the guide will be updated.

All Cisco Wireless Phone models are covered in these guides:

- Cisco Wireless Phone 840/840S
- Cisco Wireless Phone 860/860S

Note Both Cisco Wireless Phone 860 Series and 840 Series running Firmware Release 1.1 are compatible with SDK 2.4.

The Cisco Library

To use Cisco-specific APIs in your Android project, you must include the Cisco libraries in your project: `com.spectralink.sdk.jar`.

As the Cisco API changes over time, such as adding new capabilities, we will release new versions of its library. A developer should ensure the `com.spectralink.sdk.jar` file that is included in an Android project corresponds to the Cisco API version the developer intends to use (for example 2.4).

Cisco Libraries in Android Studio

The following steps describe one method for using the API in a project for Android Studio. This process is not unique to our API, but depending on your project's complexity, few more steps are required. Refer to the internet for additional information. There are likely several ways to do this so these are guidelines and not hard-and-fast rules.

Note Trying to use Cisco APIs without inclusion of the Cisco libraries cause compiler, linker, or run-time errors.

1. Add the com.spectralink.sdk.jar file to the folder app/libs within your app's project.
2. Open the application build.gradle (Module: app) and under dependencies, add: *implementation files('libs/com.spectralink.sdk.jar').*
3. Sync project and use.

Barcode API

The barcode API allows Android applications (activities and services) to receive scanned barcode data on Cisco Wireless Phone models with an integrated 1D/2D barcode reader (9x53). Applications can also enable and disable the barcode reader to prevent an accidental barcode key press from powering-on the illuminating LED in the barcode module.

- Allow multiple apps or services to receive barcode data.
- Introduce API to disable & enable the barcode scanner.
- Introduce API to determine if barcode scanner is present on device.

Usually, EMM configures barcode scanner and symbologies. Device can also be to configure them.

Supported Symbologies

The following symbologies are supported:

Aztec	Codabar
CCA EAN-128	Code 11
CCA EAN-13	Code 128

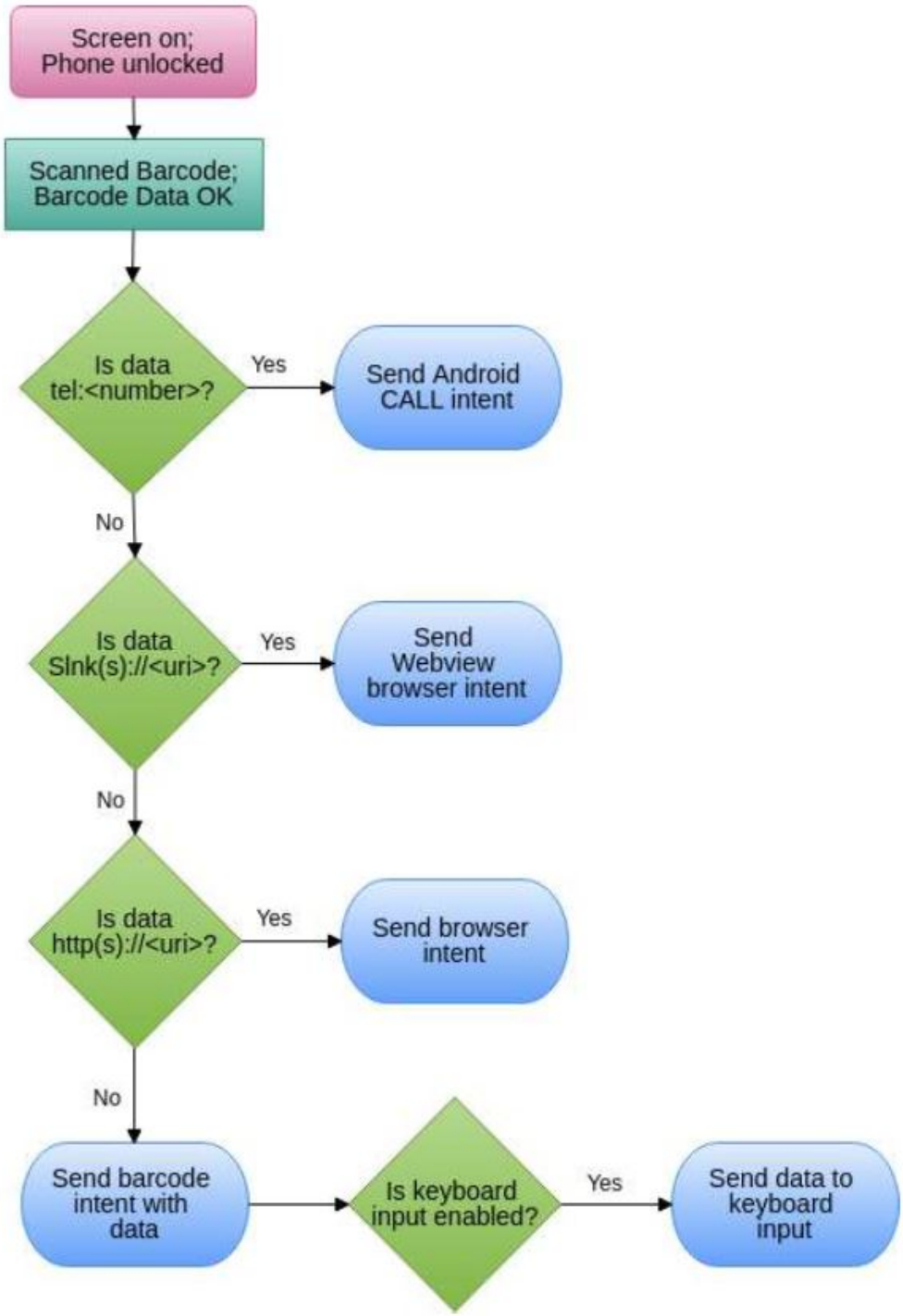
CCA EAN-8	Code 32
CCA GS1 DataBar	Code 39 Full ASCII
Expanded	Code 39 Trioptic
CCA GS1 DataBar Limited	Code 93
CCA GS1 DataBar-14	DataMatrix
CCA UPC-A	Discrete (Standard) 2 of 5
CCA UPC-E	EAN-128
CCB EAN-128	EAN-13
CCB EAN-13	EAN-13 + 2 Supplemental
CCB EAN-8	EAN-13 + 5 supplemental
CCB GS1 DataBar	EAN-8
Expanded	EAN-8 + 2 Supplemental
CCB GS1 DataBar Limited	EAN-8 + 5 supplemental
CCB GS1 DataBar-14	GS1 DataBar Expanded
CCB UPC-A	GS1 DataBar Limited
CCB UPC-E	GS1 DataBar-14
CCC EAN-128	Han Xin

The following symbologies are supported:

1D:	**2D:**
CIP 128	Australian Post
UPC-E1	British Post Office
UPC-D	Canada Post
ISMN	Codablock A
ISSN	Codablock F
	Code 16k
	Dutch Post
	Infomail
	Japan Post

Barcode Data Flow

The flow diagram shows how scanned data will be processed by the Cisco barcode service.



Barcode API

com.spectralink.barcode.lib

Class BarcodeManager

java.lang.Object

com.spectralink.barcode.lib.BarcodeManager

public class BarcodeManager

extends java.lang.Object

Table 1: Field summary

Field	Description
`static java.lang.String`	`static java.lang.String` This string can be used as the intent filter to receive scanned barcode data.
`static java.lang.String`	`SCAN_DATA_EXTRA` This is the key used to retrieve the barcode data from broadcasted SCAN_INTENTs.
`static java.lang.String`	`SCAN_DATA_SYMBOLOGY` This is the key used to retrieve the barcode symbology from broadcasted SCAN_INTENTs.
`static java.lang.String`	`SCAN_STATE_EXTRA` This is the key used to retrieve the barcode state from broadcasted STATE_INTENTs.
`static java.lang.String`	`STATE_BC_DISABLED` This string is passed as extra data with the barcode STATE_INTENT when barcode scanning is disabled.
`static java.lang.String`	`STATE_BC_ENABLED` This string is passed as extra data with the barcode STATE_INTENT when barcode scanning is enabled.
`static java.lang.String`	`STATE_INTENT` This string can be used as the intent filter to receive scanner state changes.
`static java.lang.String`	`STATE_KEYBOARD_DISABLED` This string is passed as extra data with the barcode STATE_INTENT when barcode keyboard input is disabled.
`static java.lang.String`	`STATE_KEYBOARD_ENABLED` This string is passed as extra data with the barcode STATE_INTENT when barcode keyboard input is enabled.

Table 2: Method summary

Method	Description
`void`	`disableBarcodeKeyboard(android.content.Context ctx)` Disables automatic keyboard input from the barcode manager.
`void`	`disableBarcodeReader(android.content.Context ctx)` Disables the use of the barcode scanner.
`void`	`doDecode()` Triggers a barcode scan. Note: this call only works on Cisco Wireless Phone R1.4 or greater.
`void`	`enableBarcodeKeyboard(android.content.Context ctx)` Enables automatic keyboard input from the barcode manager.
`void`	`enableBarcodeReader(android.content.Context ctx)` Enables the use of the barcode scanner.
`Static Barcode Manager`	`getInstance()` Gets an instance of the Barcode manager.
`boolean`	`getIsBarcodeEnabled()` Returns true if the barcode reader is enabled and false otherwise.
`boolean`	`getIsBarcodeKeyboardOn()` Returns true if the barcode keyboard input feature is enabled and false otherwise.
`boolean`	`hasBarcodeReader()` Returns true if the device has a barcode reader and false otherwise.

Barcode API Guidelines

See the Barcode API example app included in this SDK for more details. Android projects using the barcode capability must include the `com.spectralink.barcode.lib` library (contained within `com.spectralink.sdk.jar`). The library can also be done adding the following to the `manifest.xml` file.

```
<uses-library android:name="com.spectralink.barcode.lib" />
```

On Cisco Wireless Phones with barcode readers (for example 840s), a Cisco barcode system service is started during boot. The service is responsible for generating intents with barcode reader state and barcode data. If the above uses-library declaration has android:required="false" set, the developer needs to check for this to be a Cisco device before using any barcode API.

Determining if a barcode scanner is present

Applications can determine if a barcode scanner is present, either by checking device model numbers (i.e. using Android.os.Build MODEL field) which may be challenging to keep in sync with new Cisco or OEM product offerings, or by using the BarcodeManager hasBarcodeReader method, where the latter is the preferred approach.

Note The BarcodeManager instance shall exist even on devices without a barcode scanner.

```
barcodeManager = BarcodeManager.getInstance();
if(barcodeManager.hasBarcodeReader()){
// do something useful with reader
} else{
// no barcode reader on this phone.
}
```

Enabling / disabling the barcode scanner

To prevent a user accidentally illuminating the scanner's LED when pointed at someone, an app can control the scanner function using the disableBarcodeReader and enableBarcodeReader methods. The current scanner state can be identified via the BarcodeManager.STATE_INTENT and checking the extra data for STATE_BC_DISABLED or STATE_BC_ENABLED.

```
disableButton.setOnClickListener(new OnClickListener(){
@Override
public void onClick(View v) {
barcodeManager.disableBarcodeReader(v.getContext());
}
});
enableButton.setOnClickListener(new OnClickListener(){
@Override
public void onClick(View v) {
barcodeManager.enableBarcodeReader(v.getContext());
}
});
```

Receiving scanned barcode data

To receive barcode data, an application can register a broadcast receiver for the BarcodeManager.DATA_INTENT. The actual data is available in the extended data of the intent by using the String key BarcodeManager.SCAN_DATA_EXTRA. You can also get symbology by using the string key BarcodeManager.SCAN_DATA_SYMBOLOGY.

```
public class BarcodeReceiver extends BroadcastReceiver {
String mReceiverName = "";
BarcodeReceiver(String receiverName){
mReceiverName = receiverName;
}
@Override
public void onReceive(Context context, Intent intent) {
String rcvData = intent.getStringExtra(BarcodeManager.SCAN_DATA_EXTRA);
String rcvSymbology =
```

```
intent.getStringExtra(BarcodeManager.SCAN_DATA_SYMBOLOGY);
Logging.myLog(mReceiverName + " Received: " + rcvData, context);
Logging.myLog(mReceiverName + " Received Symbology: " + rcvSymbology,
context);
}
public class TestActivity extends Activity{
public void onCreate(Bundle savedInstanceState) {
<snip>
// Register activity barcode receiver.
bcReceiver = new BarcodeReceiver("BC Activity");
IntentFilter filter = new IntentFilter(BarcodeManager.DATA_INTENT);
registerReceiver(bcReceiver, filter);
<snip>
}
}
```

Enabling / disabling text input field data insertion

By default, Cisco Wireless Phone will input scanned data into a text input field if in focus. This is useful if the application does not actively interface with the barcode API to receive the data directly. However, some apps may not want this behavior, so the behavior can be disabled by an app using the disableBarcodeKeyboard and enableBarcodeKeyboard methods. The current keyboard input state can be identified via the BarcodeManager.STATE_INTENT and checking the extra data for STATE_KEYBOARD_DISABLED or STATE_KEYBOARD_DISABLED. If an application is using our API it is suggested to disable this keyboard capability.

```
testBarcode.disableBarcodeKeyboard(v.getContext());
```

Example code

Please see the example code package for the Barcode API.

Custom Intents

Cisco Wireless Phone 840S or 860S supports Custom Intents.

The example application in this SDK zip file demonstrates the usage of the custom Intents. The manifest.xml file has Intent Filters with Intent Action and Intent Categories.

The partner will need to provide the three settings to the SAM or EMM administrator.

- Intent Delivery Method
- Intent Action
- Intent Category

Those three settings collectively will enable the Barcode Service to send an Intent to the partner application using one of the following delivery methods after a scan is completed.

- Start Activity
- Start Service
- Start Foreground Service
- Send Broadcast

The custom intent will contain the data shown in the table below as String Extras.

String Extras can be obtained from the extras bundle shown below by calling “intent.getExtras()”

Key	Value (examples)	Notes
com.spectralink.Scanflex.data_string	GS18061200285	Barcode value after string Manipulation
com.spectralink.Scanflex.data_dispatch_time	1586158888809	Unix time of Scan
com.spectralink.Scanflex.label_type	Code 128	Type of Symbology Scanned

Use Cases

1. Start Activity use case

   The partner app wants to move from the MainActivity to a different activity on a successfully completed scan and send data to the new activity.

2. Send Broadcast use case

   The partner application wants to send a broadcast to a broadcastReceiver that it has already implemented, either within the same application or a different application developed by the same partner.

Button API

Buttons App User Interface

Cisco Wireless Phones contain multiple buttons not normally found on a consumer phone (Left, Right, Top, and Fingerprint (Cisco Wireless Phone 860 or 860S only), along with expected buttons (Power, Volume up and Volume down). Volume up and down buttons are available to apps via standard Android APIs, e.g. using class `KeyEvent` and keycode `KeyEvent.KEYCODE_VOLUME_DOWN`.

All buttons are configurable in the Buttons app except for the power button. If an app listens for a android intent via a button press, that android intent must be mapped to that respective physical button in the Buttons App. For example, remapping the left button from ‘barcode’ to ‘custom 1’ will now send the custom 1 intent when the left button is pressed.

The Buttons app provides a way for the user to change the way a button functions. The function follows the change. No matter which button is selected for a function, the function intent will be delivered when the new button is pressed.

The Buttons app allows you to change which programmable button does what assignable action:

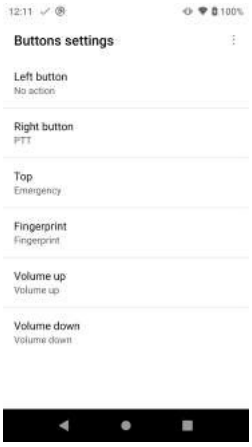
Button	Default action
LEFT button	No action
RIGHT button	PTT
Top button	Alarm
Fingerprint*	Fingerprint*
Volume up	Volume up
Volume down	Volume down
	Scanner**
	Run application
	Home key
	Back key
	Open URL
	Menu key
	Custom 1
	Custom 2
	Custom 3
	Custom 4

* Cisco Wireless Phone 860 or 860S only (on the back of the phone)

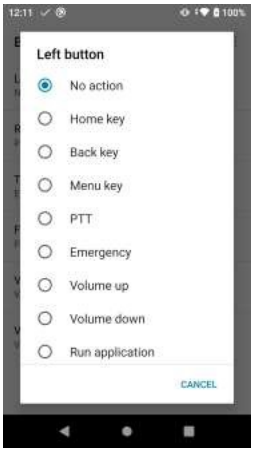
** Cisco Wireless Phone 840S or 860S only

The user simply selects the button to remap and then selects the new desired function. Not all actions are available on all buttons. Custom buttons are programmed by the system administrator.

Default



Options



Cisco Intents for Buttons App

A custom app can use the physical buttons on Cisco Wireless Phone for its own purposes. There are two steps to get the appropriate functionality:

1. The custom app must register a broadcast receiver to listen for a Cisco intent.
2. The respective Cisco intent must be mapped to that button using the Button app.

The following intents can be defined in a custom application and then registered to a receiver within that application:

Note Cisco Wireless Phones use the "Apollo" code word for intents used by all models of Cisco Wireless Phones.

Table 3: Broadcast Action Intent strings

String	Description
"com.apollo.intent.action.PTT_BUTTON"	Intent action string for PTT
"com.apollo.intent.action.PANIC_BUTTON"	Intent action string for Panic
"com.apollo.intent.action.BARCODE_BUTTON"	Intent action string for Barcode
"com.apollo.intent.action.FINGERPRINT_BUTTON"	Intent action string for Fingerprint
"com.spectralink.intent.action.CUSTOM1_BUTTON"	Intent action string for Custom 1
"com.spectralink.intent.action.CUSTOM2_BUTTON"	Intent action string for Custom 2
"com.spectralink.intent.action.CUSTOM3_BUTTON"	Intent action string for Custom 3
"com.spectralink.intent.action.CUSTOM4_BUTTON"	Intent action string for Custom 4

For each intent, the EXTRA_TEXT string provides information on the button action as follows:

Table 4: EXTRA_TEXT String

String	Description
"keypress"	When key is initially pressed
"keyrelease"	When key is released
"shortpress"	Indicates key was pressed for short duration
"longpress"	Indicates key was pressed for a duration exceeding the Android longpress threshold

Button API Guidelines

Please see the Button API example app included in this SDK for more details. The following provides a simple code snippet to detect a PTT long press.

```
public static final String PTT_BUTTON =
"com.apollo.intent.action.PTT_BUTTON";
public static final String LONGPRESS = "longpress";
public class ButtonActionReceiver extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
if(intent.getAction().equals(PTT_BUTTON)) {
Bundle b = intent.getExtras();
if(b.get(Intent.EXTRA_TEXT).equals(LONGPRESS)){
//do something cool with long key press
}
}
```

```
}
}
```

The generated intents do have dependencies based on Cisco Wireless Phone's awake or asleep state, i.e. whether the screen is on or off. The following provides an explanation of the behaviors. However, application developers should become familiar with the button intent behaviors before trying to integrate them into their app:

Phone Awake, Screen On (All buttons):

1. Button pressed -> button's respective intent generated with EXTRA_TEXT=keypress.
2. If button is held longer than Android's longpress threshold -> button's respective intent generated with EXTRA_TEXT=longpress.
3. If button is released before Android's longpress threshold -> button's respective intent generated with EXTRA_TEXT=shortpress.
4. Button released -> button's respective intent generated with EXTRA_TEXT=keyrelease.

Phone Asleep, Screen Off (button set to Scanner or Custom):

1. Button pressed -> No intent generated
2. Button released -> No intent generated

Phone Asleep, Screen Off (button set to Alarm or PTT):

1. Button pressed -> Phone wakes, NO keypress intent generated
2. If button is held longer than Android's longpress threshold -> button's respective intent generated with EXTRA_TEXT=longpress.
3. If button is released before Android's longpress threshold -> button's respective intent generated with EXTRA_TEXT=shortpress.
4. Button released -> button's respective intent generated with EXTRA_TEXT=keyrelease.

As shown above, buttons can behave differently, and behavior differs depending on phone state.

Buttons Troubleshooting

Adding Logcat messages will be helpful for identifying when and what Intents are received.

Miscellaneous

The following section provides additional useful programming information for Cisco Wireless Phone. These may not use Cisco proprietary APIs but offer useful Standard Android based hints.

Initiating a Call Using Cisco SIP Dialer

Apps may want to use the integrated Cisco SIP dialer app to initiate phone calls.. Calling can be done using the standard Android Intents, ACTION_CALL and ACTION_DIAL. See Android documentation for full details on semantics. However in general terms, ACTION_CALL initiates a call, but requires Manifest permissions, whereas ACTION_DIAL does not actually start the call nor does it require Manifest permission.

An example of using the intent is:

```
Intent callIntent = new Intent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse("tel:7203754157"));
startActivity(callIntent);
```

There are many examples on the Internet, one good reference is:http://www.mkyong.com/android/how-to-make-a-phone-call-in-android/

Google Play Services

Cisco Wireless Phone is a Google certified device and accordingly the software now includes and supports Google Mobile Services. Google Play, Google Play Services, and associated APIs are available to applications as applicable.