Spectralink Versity Smartphone Family

API Specification 2.6

Support for
Spectralink Versity 95/96/97 smartphones using Android 13

721-0058-000 Rev L
March 2025

Copyright Notice

Notice

Warranty

The *Product Warranty and Software License and Warranty* and other support documents are available at http://support.spectralink.com.

Contact Information

US Location
+1 800-775-5330
Spectralink Corporation
305 S. Arthur Avenue
Louisville, CO 80027
USA
info@spectralink.com

Denmark Location
+45 7560 2850
Spectralink Europe ApS
Bygholm Soepark 21 E Stuen
8700 Horsens
Denmark
infoemea@spectralink.com

UK Location
+44 1344 206591
Spectralink Europe ApS—UK branch
Suite B1, The Lightbox
Bracknell, Berkshire, RG12 8FB
United Kingdom
infoemea@spectralink.com

Contents

Revision History

Rev	Change Description	Author	Effectivity date
L	SDK 2.6 Revised to cover Versity A13 phones. Text and format changes for clarity. Removed V92 references. Changed *Custom* to *Programmable*. Noted that Programmable is not an option for the V97 Volume Down button.	Kelly B	
K	SDK 2.6 Revised text, graphics to include Versity 97 Series. Added A13 info to support Versity 97 Series.	Kelly B. John W	
J	SDK 2.5 Add Ch 4 Device Info.	Sue A Richard H	
H	SDK 2.4 Modify text for Versity 92 Series. No changes to code or other technical information except to clarify which model is being discussed.	Sue A Richard H	10-21-20
G	SDK 2.3 Released w/Versity 95/96 Series QR4 R1.8, Barcode service. Add Custom Intents section.	SueA VivekM	4-10-20
F	SDK 2.2 Released w/Versity 95/96 Series, QR3 R1.7 Barcode app. Add additional symbologies to list of supported symbologies. Add UPC-D and Code 16k to list of unsupported symbologies. For Barcode API add SCAN DATA SYMBOLOGY for symbology retrieval. Add sample code.	SueA VivekM	3-20-20
E	SDK 2.1 Delete CC-AB and CC-C from 1D list as inapplicable. Update Button app for new Custom buttons, etc.	Sue A Phil W	7-19-19
D	SDK 2.1 Minor typo edits. Revised barcode flow diagram. Corrections to Buttons app chapter.	Steve M	6-19-19
C	SDK 2.1 Updated to 2.1 API specification, extensive edits.	Steve M	4-29-19
B	SDK 2.0 General review for Versity GA. Release for AIMS partners only.	Sue A	10-1-18
A	SDK 2.0 API Spec for Versity leveraging 1.3 API Spec for PIVOT. BETA version. Not for broad release.	Marcus M	6-1-18

Chapter 1: About This Specification

This document specifies Spectralink's Versity Application Programming Interfaces (APIs) which expose Versity platform capabilities not available through standard Android OS (AOSP) application APIs, such as access to scanned barcode data etc.

The specification is written for native Android application (app) developers and assumes Android application programming competency. For technical development questions, contact: aims@spectralink.com.

As additional API platform capabilities become available or as existing APIs are revised, the API version shall change, and this document will be updated.

Versity Series 95/96/97 running Android 13 are covered in this document.

ADMIN TIP
Software versions

This document covers the following Spectralink Versity models and software releases:

- Versity 95/96 Series running Versity 3.1
- Versity 97 Series running Versity 13.3

Although this document covers Versity devices running Android 13, all Versity models are compatible with SDK 2.6, including Versity phones running Android 10. Refer to the previous revision of this document, Rev. K, for specific information regarding those devices.

The Spectralink Library

To use Spectralink specific APIs in your Android project, you need to include the Spectralink libraries in your project: *com.spectralink.sdk.jar*.

As the Spectralink API changes over time, such as adding new capabilities, Spectralink will release new versions of its library. A developer should ensure the `com.spectralink.sdk.jar` file included in an Android project corresponds to the Spectralink API version the developer intends to use (e.g. 2.6).

Using Spectralink's Libraries in Android Studio

The following steps describe one method for using our API in a project for Android Studio. There's more than one way to do this, and this process is not unique to our API. Depending on your project's complexity, additional steps may be necessary. Additional information on this process can be found online.

ADMIN TIP

Use Spectralink libraries

Trying to use Spectralink APIs without inclusion of the Spectralink library will cause compiler, linker, or run-time errors.

(NOTE: An earlier SDK release included the file *com.spectralink.barcodelib.jar*. This file is no longer needed and is not a part of the current SDK.)

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

Chapter 2: Barcode API

The barcode API allows Android applications (activities and services) to receive scanned barcode data on Versity smartphone models with an integrated 1D/2D barcode reader (9x53). Applications can also enable and disable the barcode reader, which can be used to prevent an accidental barcode keypress 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

The barcode scanner and symbologies are configurable by the device but are usually configured by the SAM server or an EMM.

Supported Symbologies

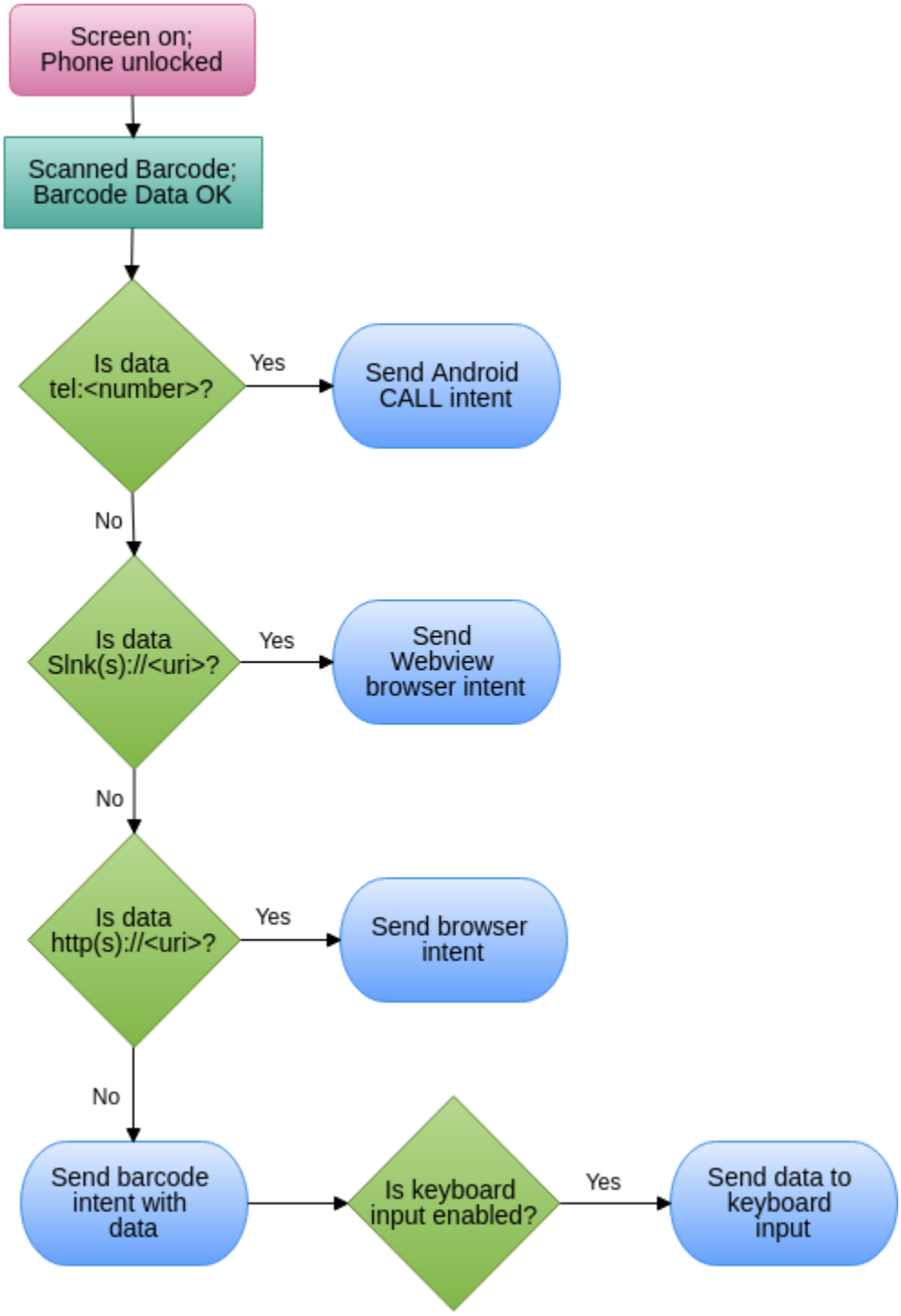
Aztec
CCA EAN-128
CCA EAN-13
CCA EAN-8
CCA GS1 DataBar Expanded
CCA GS1 DataBar Limited
CCA GS1 DataBar-14
CCA UPC-A
CCA UPC-E
CCB EAN-128
CCB EAN-13
CCB EAN-8
CCB GS1 DataBar Expanded
CCB GS1 DataBar Limited
CCB GS1 DataBar-14
CCB UPC-A
CCB UPC-E
CCC EAN-128
Codabar
Code 11
Code 128
Code 32
Code 39 Full ASCII
Code 39 Trioptic
Code 93
DataMatrix
Discrete (Standard) 2 of 5
EAN-128
EAN-13
EAN-13 + 2 Supplemental
EAN-13 + 5 supplemental
EAN-8
EAN-8 + 2 Supplemental
EAN-8 + 5 supplemental
GS1 DataBar Expanded
GS1 DataBar Limited
GS1 DataBar-14
Han Xin
Interleaved 2 of 5
ISBT-128
ISBT-128 Con
Macro Micro PDF
Macro PDF
Macro QR
Matrix 2 of 5
Micro PDF
Micro QR
MSI
PDF-417
QR Code
UPC-A
UPC-A + 2 Supplemental
UPC-A + 5 supplemental
UPC-E0
UPC-E0 + 2 Supplemental
UPC-E0 + 5 supplemental

The following symbologies are not supported

1D:	**2D**:	
CIP 128	Australian Post	MaxiCode
UPC-E1	British Post Office	MaxiCode Mode 0
UPC-D	Canada Post	Irregular PDF
ISMN	Codablock A	Planet Postal
ISSN	Codablock F	Postnet
	Code 16k	QR Code Model 1
	Dutch Post	Sweden Post
	Infomail	TLC 39
	Japan Post	

Barcode data flow

The following diagram shows how scanned data will be processed by the Spectralink 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
```

Field summary	
`static java.lang.String`	`DATA_INTENT` 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.

Method summary	
`void`	`disableBarcodeKeyboard(android.content.Context ctx)` Disables automatic keyboard input from the barcode manager.

Method summary	
`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 Versity 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(android.content.Context appContext, BarcodeServiceStatusCallback callback)` 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 use guidelines

Please see the Barcode API example app included in this SDK for more details. Android projects using the barcode capability need to include the `com.spectralink.barcode.lib` library (contained within `com.spectralink.sdk.jar`). (See Using Spectralink's Libraries in Android Studio.)

On Versity smartphones with barcode readers (9553, 9653, and 9753 models only), a Spectralink barcode system service is started during boot. The service is responsible for generating intents with barcode reader state and barcode data.

Access to BarcodeManager object

Applications can obtain a `BarcodeManager` object by calling the static method `getInstance()`.The application context parameter is used to call `bindService()` which is an asynchronous system call to connect to the Barcode Service. A `BarcodeManager` object will be immediately returned to the caller, but the object will not be ready to be used until the `BarcodeServiceStatusCallback` is called with the status `BarcodeManager.BarcodeServiceStatusCallback.CONNECTED`.

```
BarcodeManager startBarcodeService() {
    // Use the Deprecated version of getInstance() with no Application Context
    // when running on Android 10
    if (Build.VERSION.SDK_INT < 33) {
        // Android 10
        mBarcodeManager = BarcodeManager.getInstance();
    } else {
        // Android 11+
        jwarner
        mStatusCallback = new BarcodeManager.BarcodeServiceStatusCallback() {
            jwarner
            @Override
            public void statusUpdate(int i) {
                if(BarcodeManager.BarcodeServiceStatusCallback.CONNECTED == i) {
                    mBarcodeManager.enableBarcodeReader(getApplicationContext());
                }
                if(BarcodeManager.BarcodeServiceStatusCallback.CONNECTED == i ||
                   BarcodeManager.BarcodeServiceStatusCallback.DISCONNECTED == i ) {
                    initSwitches();
                }
            }
        };
        // The newer (Android 11+ compatible) version of
        // getInstance() attempts to bind to the remote barcode service
        // this connection may take some time and completes asynchronously
        mBarcodeManager = BarcodeManager.getInstance(getApplicationContext(), mStatusCallback);
    }
    return mBarcodeManager;
}
```

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 Spectralink 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.

```
if(mBarcodeManager.hasBarcodeReader()) {
    // do something useful with reader
} else {
    // no barcode reader on this device
}
```

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 TestActivity extends Activity{

    public void onCreate(Bundle savedInstanceState) {
        // snip
        mBarcodeDataReceiver = new BarcodeDataReceiver();
        mDataFilter = new IntentFilter(BarcodeManager.DATA_INTENT);
        registerReceiver(mBarcodeDataReceiver, mDataFilter,
            "com.spectralink.BARCODE_DATA", null);
        // snip
    }


public static class BarcodeDataReceiver extends BroadcastReceiver {
    @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);
    }
}
```

NOTE: Add the following uses-permission to the application AndroidManifest.xml:

- `com.spectralink.BARCODE_ENABLE_DISABLE` is a permission to use with `registerReceiver()` to receive intents when the Barcode Service is enabled or disabled.
- `com.spectralink.DATA_INTENT` is a permission to use with `registerReceiver()` to receive intents when a barcode is scanned to obtain the barcode data.

```
<uses-permission android:name="com.spectralink.BARCODE_ENABLE_DISABLE" />
<uses-permission android:name="com.spectralink.BARCODE_DATA" />
```

Enabling / disabling text input field data insertion

By default, Versity 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.

```
public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
    if (isChecked) {
        // The toggle is enabled
        if ((mBarcodeManager != null) && (mBarcodeManager.hasBarcodeReader()))
            mBarcodeManager.enableBarcodeKeyboard(buttonView.getContext());
    } else {
        // The toggle is disabled
        if ((mBarcodeManager != null) && (mBarcodeManager.hasBarcodeReader()))
            mBarcodeManager.disableBarcodeKeyboard(buttonView.getContext());
    }
}
```

Example code

Please see the example code package for the Barcode API.

Programmable Intents

Barcode service supports Programmable Intents.

NOTE
Programmable intents are backwards compatible

This update is backwards compatible. Pre-Versity 95/96 Series R1.8, broadcasts sent with raw barcode data will still be sent.

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 AIMS 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 AIMs 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 AIMS 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

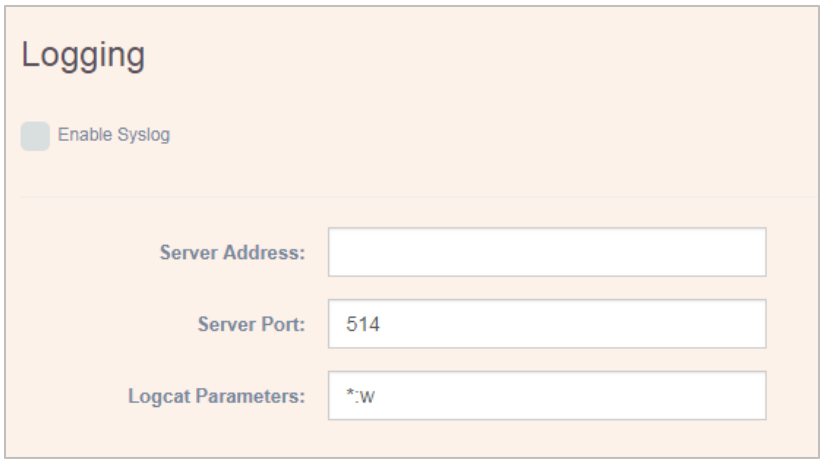
   The AIMS partner application wants to send a broadcast to a `broadcastReceiver` that it has already implemented, either within the same application or a different application developed by the same AIMs partner.

Barcode Troubleshooting

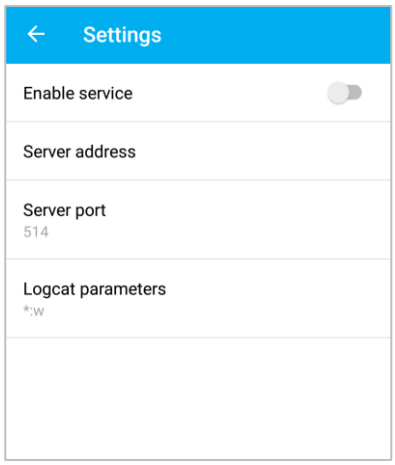
A useful mechanism for debugging barcode scanning is to enable the barcode tags (“barcode”, “BarcodeManager”, “BarcodeService”) in the Syslog Logs. This can be done via SAM or by

using the Logging app. Point the smartphone at a syslog server, then set the tags in the Logcat Parameters field Specifically use **BarcodeImpl:V** to get more logcat from barcode. See https://developer.android.com/studio/command-line/logcat for different levels.

SAM Logging app



Versity Logging app



At the verbose level, “barcode” dumps a lot of information, however upon a successful scan it lists the length of characters scanned. You may be able to lower the log level from verbose to reduce undesired information. Below is an example debug output:

May 20 17:13:32	192.168.1.116	local0.debug	May 20 07:13:33 192.168.1.116 barcode [2230]: 1400627613222|barcode |7|00| Scan Key Up
May 20 17:13:29	192.168.1.116	local0.info	May 20 07:13:30 192.168.1.116 battery_level[2534]: 1400627610271|battery_level|6|00| [76,3861,340]
May 20 17:13:29	192.168.1.116	local0.info	May 20 07:13:29 192.168.1.116 barcode [2229]: 1400627609925|barcode |6|00| [BarCode]: Read
May 20 17:13:29	192.168.1.116	local0.debug	May 20 07:13:29 192.168.1.116 barcode [2229]: 1400627609919|barcode |7|00| [ISCP]: resultBufLen=255, rc=263
May 20 17:13:29	192.168.1.116	local0.debug	May 20 07:13:29 192.168.1.116 barcode [2230]: 1400627609703|barcode |7|00| Scan Key Down

Chapter 3: Button API

Buttons App User Interface

Versity smartphones contain multiple buttons not typically found on consumer phones (Left, Right, Top, Fingerprint *[95/96 Series]*, Rear *[97 Series]*), Alarm) along with common buttons (Power, Volume up, 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_UP`.

All buttons are configurable in the Buttons app except for the power button. If an app listens for a Spectralink intent via a button press, that Spectralink intent must be mapped to that respective physical button in the Buttons app. For example, remapping the left button from *Scanner* to *Programmable* will now send the Programmable 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	Scanner*
Right button	PTT *(95/96)*; Home key *(97)*
Top button	Alarm
Fingerprint *(95/96)*	Fingerprint *(95/96)*
Rear button *(97)*	No action *(97)*
Volume up	Volume up
Volume down	Volume down

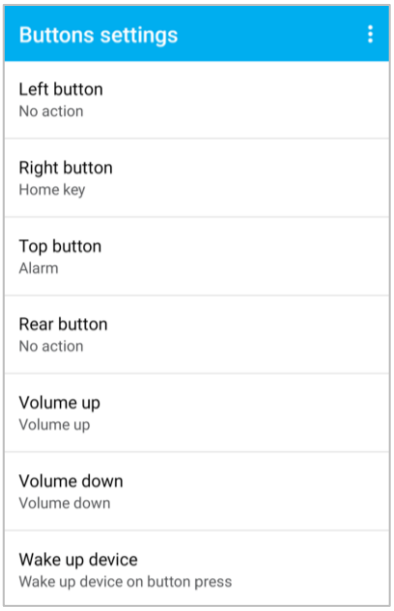
Additional actions available
Run application
Home key
Back key
Open URL
Menu key
Programmable**
Custom 1
Custom 2
Custom 3
Custom 4

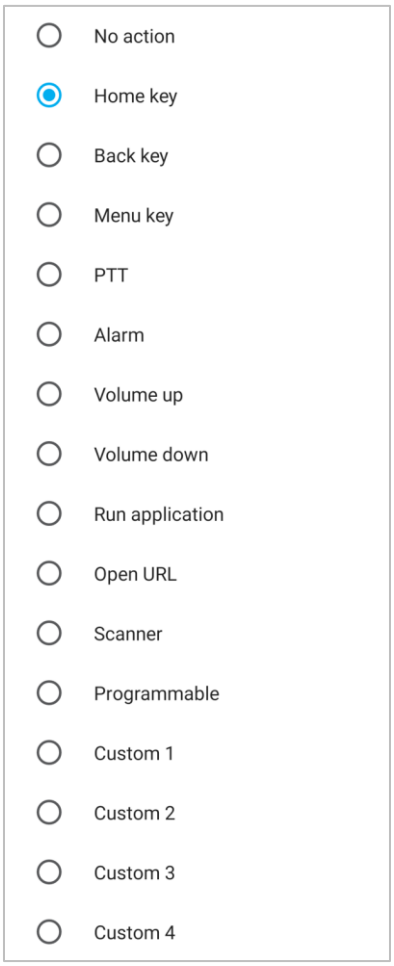
** Only on the Versity models with a barcode scanner: 9553/9653/9753*

** Programmable *is not an available action for the* Volume down *button on the 97 Series.*

The user 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. See the *Spectralink Applications Administration Guide* for detailed information.

At right: the Buttons settings on a 97 Series phone, and the action list that appears when "Right button" is selected.





Spectralink Intents for Buttons App

A custom app can use the physical buttons on Versity 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 Spectralink intent.
2. The respective Spectralink 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:

Admin Tip
The "Apollo" intent is applicable to all Versity models

Versity smartphones use the "Apollo" code word for intents used by all models of Versity smartphones.

Broadcast Action Intent strings	
"com.apollo.intent.action.PTT_BUTTON"	Intent action string for PTT *(95/96 Series only)*
"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 *(95/96 Series only)*
"com.apollo.intent.action.BACK_BUTTON"	Intent action string for Rear *(97 Series only)*
"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:

EXTRA_TEXT String	
"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 use 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 Versity'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 identifying when and what Intents are received.

Chapter 4: Device Info

Starting in Android 10 Google made privacy changes that prevents non-system applications from accessing device serial numbers and Wi-Fi MAC addresses. This may create some difficulty for application partners who would want these values. Accordingly, Spectralink has expanded our API to allow access to both. An example of API usage is provided in sdkExample.

DeviceInfo class

The DeviceInfo API is available on Versity devices running Android 13. No additional permissions are necessary in the app AndroidManifest.xml.

```
import com.spectralink.slnkdevicesettings.lib.DeviceInfo;
//...
String serialNumber = DeviceInfo.getSerialNumber()
String wifiMac = DeviceInfo.getWiFiMacAddress();
//…
```

DeviceInfoBackwardsCompat class

The `DeviceInfoBackwardsCompat` class will return the correct serial number and Wi-Fi MAC address. To use this class, add the following to your AndroidManifest.xml:

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
```

In your java class where you will retrieve the serial number and/or Wi-Fi MAC address:

```
import com.spectralink.slnkdevicesettings.lib.DeviceInfoBackwardsCompat;
//...
String serialNumber = DeviceInfoBackwardsCompat.getSerialNumber()
String wifiMac = DeviceInfoBackwardsCompat.getWiFiMacAddress(mContext);
//...
```

Chapter 5: Miscellaneous

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

Initiating a Call via Spectralink SIP Dialer

Apps may want to use the integrated Spectralink SIP dialer app to initiate phone calls (Biz Phone). This can be done using the standard Android Intents, `ACTION_CALL` and `ACTION_DIAL`. Refer to Android documentation for full-details on semantics etc. However in general terms, `ACTION_CALL` will actually initiate 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 lots of examples on the Internet, one good reference is:
http://www.mkyong.com/android/how-to-make-a-phone-call-in-android/

TelephonyManager Class

Considering that Spectralink has a LTE model of Versity (Versity 96 Series), we are considering the necessity to support this class. Please contact the AIMS manager to discuss your case.

Google Play Services

Versity is a Google certified device and accordingly the software now includes and/or supports Google Mobile Services. This means Google Play, Google Play Services and associated APIs are available to applications as applicable.

****END OF DOCUMENT***