IBScanUltimate Getting Started Guide

Version 1.9.7 (April 27, 2017)

Copyright @2011-2017, Integrated Biometrics LLC. All Rights Reserved

Table of Contents

REVISION HISTORY 3

1. SDK CONTENTS 4

1.1. Packages 4

1.2. Scanner Model Support 4

1.3. Further Information 7

2. INSTALLATION GUIDES 8

2.1. Windows Installation Guide 8

Installing the SDK 8

SDK Contents 8

Sample Applications 8

2.2. Linux (Intel) Installation Guide 10

Opening the SDK 10

SDK Contents 10

Installing the IBScanUltimate Library 10

Compiling and Running the Sample Application 11

2.3. Linux (ARM) Installation Guide 13

Opening the SDK 13

SDK Contents 13

Dependencies 14

Compiling the Sample Application 15

Installing the IBScanUltimate Library 15

2.4. Android Installation Guide 17

SDK Contents 17

Setting up an Eclipse Project with IBScanUltimate 18

Opening and Running the Example Project 19

The IBScanUltimate Java Application Interface 20

The IBScanUltimate C/C++ Application Interface 21

Using USB Devices on Android 21

3. FINGERPRINT CAPTURE GUIDE 24

3.1. Flat Print Captures 24

3.2. Rolled Print Captures.....25

Revision History

Date	Author	Remarks
2013/10	BAN	Corrected typographical error in Linux (ARM) getting started section.

1. SDK Contents

1.1. Packages

The full IBScanUltimate SDK consists of six separate packages for individual operating systems. The core of each package is the IBScanUltimate library (distributed as a DLL on Windows and shared object libraries on other platforms) that provides an API for capturing images from IB scanners. One or more sample applications in each package demonstrate the use of the library. A Java interface is also available, through a JNI bridge (distributed as a DLL on Windows) and a Java-language wrapper (distributed as a JAR).

The following are the provided packages, where “x.y.z” is the version number such as “1.6.7”:

- *IBScanUltimateSDK Setup x.y.z.exe* is the installer for all 32-bit Windows platforms, including XP, Vista, Windows 7, and Windows 8. Sample applications are provided for a variety of languages, among them C/C++, C#, VB, and Java.
- *IBScanUltimateSDK(x64) Setup x.y.z.exe* is the installer for all 64-bit Windows platforms, including XP, Vista, Windows 7, and Windows 8. Sample applications are provided for a variety of languages, among them C/C++, C#, VB, and Java.
- *IBScanUltimate_x86_x.y.z.tgz* is the release for 32-bit (x86) Linux distributions; the provided library should work on 2.6 kernels and later. A sample application is provided for C/C++.
- *IBScanUltimate_x64_x.y.z.tgz* is the release for 64-bit (x64) Linux distributions; the provided library should work on 2.6 kernels and later. A sample application is provided for C/C++.
- *IBScanUltimate_armv7a_x.y.z.tgz* is the release for ARMv7-A (Cortex-A5, Cortex-A8, Cortex-A9) Linux distributions; the provided library should work on 2.6 kernels and later. A sample application is provided for C/C++.
- *IBScanUltimate_Android_x.y.z.tar.gz* is the release for Android; the provided library should work on v4.0.0 and later Android versions. A sample app is provided.

1.2. Scanner Model Support

IBScanUltimate supports the following Integrated Biometrics fingerprint sensors:

- Watson
- Watson Mini
- Sherlock
- Columbo
- Curve and eCurve

Feature		Watson	Watson Mini	Sherlock	Columbo	Curve	eCurve
Sensor type		Light-emitting sensor (LES) CMOS CIS camera	Light-emitting sensor (LES) CMOS CIS camera	Light-emitting sensor (LES) TFT camera	Light-emitting sensor (LES) CMOS CIS camera	Light-emitting sensor (LES) CMOS CIS camera	Light-emitting sensor (LES) CMOS CIS camera
Resolution		500-pixels/inch	500-pixels/inch	500-pixels/inch	500-pixels/inch	500-pixels/inch	500-pixels/inch
Active sensing area		1.6" x 1.5"	1.6" x 1.5"	1.6" x 1.5"	0.8" x 1.0"	0.6" x 0.7" (15-mm x 18-mm)	0.6" x 0.7" (15-mm x 18-mm)
Image size		800- x 750-pixels	800- x 750-pixels	800- x 750-pixels	400- x 500-pixels	288- x 352-pixels	288- x 352-pixels
Scanner physical size		63-mm x 70-mm x 32-mm	60-mm x 61-mm x 34-mm	60-mm x 64-mm x 14-mm	39-mm x 46.5-mm x 27.5-mm	69.7-mm (diam) x 33.2-mm (height)	21-mm x 29.5-mm x 21.5-mm
USB board physical size		62-mm x 55-mm	N/A	N/A	52-mm x 45-mm	N/A	75-mm x 45-mm
Interface		USB 2.0	USB 2.0	USB 2.0	USB 2.0	USB 2.0	USB 2.0
Frame rate		15 frames/sec	15-frames/sec	10-frames/sec	7-frames/sec	7-frames/sec	7-frames/sec
Certifications		Appendix F Mobile ID IQS SAP45 2-finger and roll certified	Appendix F Mobile ID IQS SAP45 2-finger and roll certified	Appendix F Mobile ID IQS SAP45 2-finger and roll certified	PIV Mobile ID IQS SAP30	Non-certified	Non-certified

Feature	Watson	Watson Mini	Sherlock	Columbo	Curve	eCurve
Fingerprint capture types	Single-finger flat Two-finger flat Single-finger rolled	Single-finger flat Two-finger flat Single-finger rolled	Single-finger flat Two-finger flat Single-finger rolled	Single-finger flat	Single-finger flat	Single-finger flat
LEDs*	Red/green	Not supported	Not supported	Not supported	Supported	Not supported
Touch sensor**	Supported	Not supported	Not supported	Not supported	Not supported	Not supported
LE power operation***	Supported	Supported	Not supported	Supported	Supported****	Not supported
Operating systems	Windows, Linux, Android	Windows, Linux, Android	Windows, Linux, Android	Windows, Linux, Android	Windows, Linux, Android	Windows, Linux, Android

* If unsupported, the IBSU_GetLEDs() and IBSU_SetLEDs() API functions are not supported
** If unsupported, the IBSU_IsTouchedFinger() API function is not supported
*** If unsupported, the IBSU_GetLEOperationMode() and IBSU_SetOperationMode() functions are not supported
****Controls the touch sensor input

All models are supported by each platform's release. Table 1 summarizes the features of the individual sensors, and Table 2 summarizes the features of IBScanUltimate supported for each one.

1.3. Further Information

For more information about the C/C++ API of the IBScanUltimate library, please refer to *IBScanUltimate API Manual For C.pdf*.

For more information about the Java API (including the Android API) of the IBScanUltimate library, please refer to *IBScanUltimate API Manual for Java (and Android).pdf*.

For the version history of IBScanUltimate, please refer to *IBScanUltimate Version History.pdf*.

For more information about a scanner model, please refer to its hardware manual.

2. Installation Guides

Each of the following sections steps through the installation procedure on one platform and summarizes the contents that will be installed with the SDK.

2.1. Windows Installation Guide

Installing the SDK

Execute (usually by double-clicking) the Windows installer, which will be

IBScanUltimate SDK Setup x.y.z.exe

for the 32-bit content and

IBScanUltimate(x64) SDK Setup.x.y.z.exe

for the 64-bit content. Progress through the automatic installer; we recommend accepting default values whenever prompted.

SDK Contents

The SDK contains the library and sample application needed to start developing a Windows application that interfaces with an IB scanner. The material is separated into several directories, including the following:

- The */Bin* directory contains compiled sample applications, the IBScanUltimate DLL (*IBScanUltimate.dll*), and the DLLs and JARs for the Java interface (*IBScanCommon.jar*, *IBScanUltimate.jar*, *IBScanUltimateJNI.dll*).
- The */Driver* directory contains the drivers for IB scanners. These should have been installed with the SDK installer.
- The */Include* directory contains the include files for the C interface of IBScanUltimate.
- The */Lib* directory contains the compiled IBScanUltimate library file for linking.
- The */Sample sources* directory contains the source for the sample applications. The applications are separated by language.

Sample Applications

The compiled sample applications should appear within the program menu, which will also link to the folder containing the source for the applications. The following applications are provided (listed by the names that appear in the program menu):

- A series of basic samples for several different languages, each offering the same essential GUI and controls for basic scanner operations:
  - IBScanUltimate_SampleForC# is the basic sample for C#.
  - IBScanUltimate_SampleForDelphi is the basic sample for Delphi.

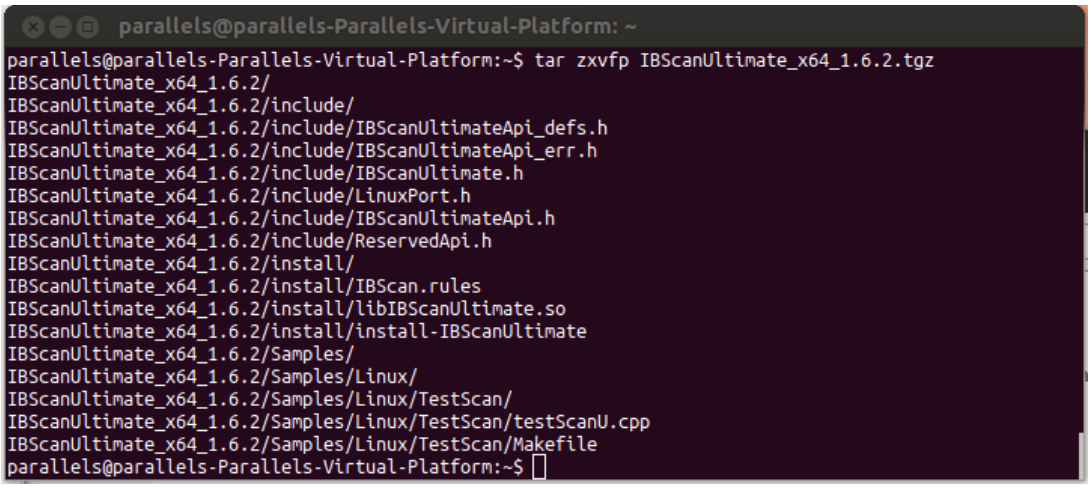
  - IBScanUltimate_SampleForVBNet is the basic sample for VB.Net.
  - IBScanUltimate_SampleForVisualBasic is the basic sample for VB6.
  - IBScanUltimate_SampleForVC is the basic sample for C/C++.
- IBSU_FunctionTester (a sample for C/C++) enumerates the API functions presented by IBScanUltimate to allow for fine-grained testing of individual library features and presentation of results.
- IBSU_FunctionTesterForJava duplicates IBSU_FunctionTester for the Java API.
- IBSU_NewFunctionTester (a sample for C/C++) improves IBSU_FunctionTester with an updated GUI and more discretion over execution of library API functions.
- IBSU_NonCallbackSample (a sample for C/C++) duplicates IBScanUltimate_SampleForVC; however, instead of relying on callbacks, the application polls library API functions to obtain status and results.
- IBSU_TenScanSample (a sample for C/C++) steps through a sequence of captures for a 10-print scan.

2.2. Linux (Intel) Installation Guide

Opening the SDK

Please copy the file *IBScanUltimate_x86_x.y.z.tgz* or *IBScanUltimate_x64_x.y.z.tgz* to your development system. Extract the contents with following command (substituting `x64` for `x86`, if necessary, and the version number for `x.y.z`):

```
# tar zxvfp IBScanUltimate_x86_x.y.z.tgz
```



SDK Contents

The SDK contains the library and sample application needed to start developing a Linux application that interfaces with an IB scanner. The material is separated into three directories:

The */include* directory contains the include files for the C interface of IBScanUltimate.

The */install* directory contains a script (*Install-IBScanUltimate*) to install the IBScanUltimate library as well as the library itself (*libIBScanUltimate.so*).

The */Samples* directory contains source code for the sample application.

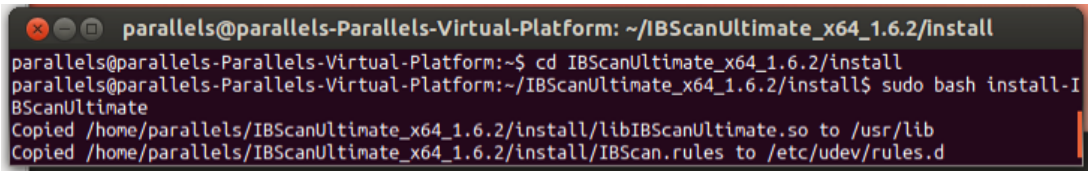
Installing the IBScanUltimate Library

First, plug in the IB USB device to your USB host port.

Second, install the IB driver library on your Linux system with commands (substituting `x64` for `x86`, if necessary, and the version number for `x.y.z`):

```
# cd IBScanUltimate_x86_x.y.z/install
# sudo ./install-IBScanUltimate
```

Some Linux distributions (like Ubuntu) need root access, obtained with the `sudo` command, to install the driver library.



Compiling and Running the Sample Application

Now you can compile and run our sample program:

```
# cd ../Samples/Linux/TestScan
# make
# sudo ./testScanU
```

Depending on the permissions you have granted to the user for accessing USB devices, you may need to grant root access to run the sample program.

```
parallels@parallels-Parallels-Virtual-Platform: ~/IBScanUltimate_x64_1.6.2/Samples/Li
parallels@parallels-Parallels-Virtual-Platform:~/IBScanUltimate_x64_1.6.2/install$ cd ../Samples/Linux
/TestScan
parallels@parallels-Parallels-Virtual-Platform:~/IBScanUltimate_x64_1.6.2/Samples/Linux/TestScan$ make
gcc -Wl,--no-as-needed -lstdc++  -fPIC -DBSD -D__linux__ -O2 -I ../../../include -m64 -Wall -lIBScanUl
timate -o testScanU testScanU.cpp
parallels@parallels-Parallels-Virtual-Platform:~/IBScanUltimate_x64_1.6.2/Samples/Linux/TestScan$ sudo
 ./testScanU
IBScanUltimate Product version: 1.6.2.0, File version: 1.6.2.0
Found 1 devices attached
WATSON_0.14.1 S/N(1208-00015) on USB

Ready.   Enter choice:
        1. Start capture for flat single finger.
        2. Start capture for flat two fingers.
        3. Start capture for rolling single finger.
        4. Abort Capture
        5. End program
:==>1

Initializing device... 0%

Initializing device... 10%

Initializing device... 17%

Initializing device... 24%

Initializing device... 32%

Initializing device... 39%

Initializing device... 46%

Initializing device... 54%

Initializing device... 61%

Initializing device... 68%

Initializing device... 76%

Initializing device... 83%

Initializing device... 100%
 Setting up for scan with callback...Displayed 'C'=Image callback.

Ready.   Enter choice:
        1. Start capture for flat single finger.
        2. Start capture for flat two fingers.
        3. Start capture for rolling single finger.
        4. Abort Capture
        5. End program
:
-- Finger count changed -- Device= 0, State= NON-FINGER
CCCCCCCCCCCCCCCCCCCCCCCCCCCCC
-- Finger count changed -- Device= 0, State= FINGER_COUNT_OK
CCCCCCCCC
Stopped. 14.5 frames per second

Flat single finger Image acquisition complete
Saving image...
NFIQ score is 3


Press enter!
???
```

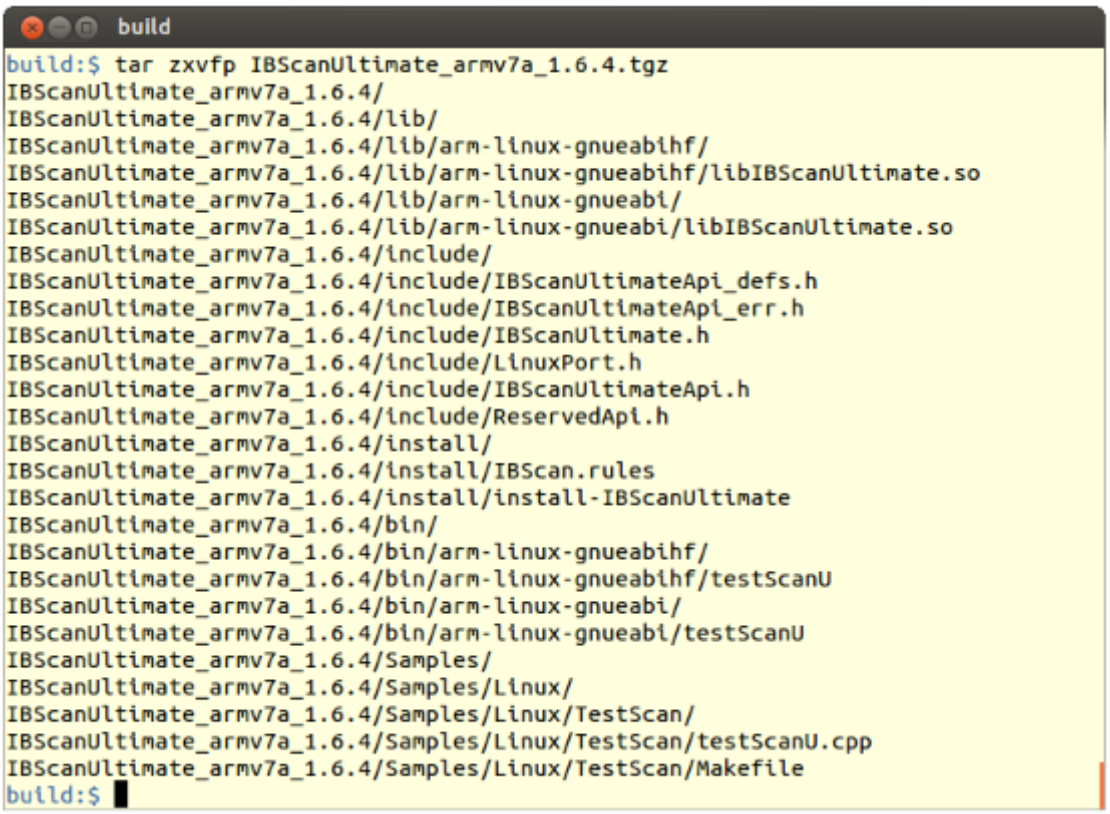
2.3. Linux (ARM) Installation Guide

Opening the SDK

Copy the file *IBScanUltimate_armv7a_x.y.z.tgz* to either your host (your desktop Linux computer or VM) or target (the ARM Linux device). Since the sample application may need to be rebuilt, the choice will depend on the location of your build tools (either a cross-compiler on the host or native tools on the target). (In the sample commands below, the build and target systems are differentiated by their prompts—`Build:$` versus `Target:$`—even though these will coincide if you build natively.)

Extract the contents with following command (substituting the version number for `x.y.z`):

```
Build:$ tar zxvfp IBScanUltimate_armv7a_x.y.z.tgz
```



SDK Contents

The SDK contains the library and sample application needed to start developing a Linux application that interfaces with an IB scanner. The material is separated into five directories:

The */bin* directory contains versions of the compiled sample application (*testScanU*), separated into directories by ABI. These binaries have been compiled dynamically against the dependent libraries under a cross-compiler and may not execute on all targets. You may need to recompile the application

with the proper toolchain.

The */include* directory contains the include files for the C interface of IBScanUltimate.

The */install* directory contains a script (*Install-IBScanUltimate*) to install the IBScanUltimate library.

The */lib* directory contains versions of the IBScanUltimate library (*libIBScanUltimate.so*) separated into directories by ABI.

The */Samples* directory contains source code for the sample application.

The distribution has separate binaries and libraries for two ABIs:

The arm-linux-gnueabi version should be installed on platforms with system libraries and applications built for the "soft" floating-point ABI, equivalent to the GCC flags

```
-mthumb -mfloat-abi=softfp -march=armv7-a
```

The arm-linux-gnueabihf version should be installed on platforms with system libraries and applications built for the "hard" floating-point ABI, equivalent to the GCC flags

```
-mthumb -mfloat-abi=hard -march=armv7-a
```

Dependencies

The IBScanUltimate library requires that libusb (both libusb-1.0 and the compatibility layer, libubs-0.1) and libudev be installed on the target. On Ubuntu, you would use `apt-get`:

```
Target:$ sudo apt-get install libusb-0.1-4
Target:$ sudo apt-get install libudev0
```

On Angstrom, you would use `opkg`:

```
Target:$ opkg install libusb-0.1-4
Target:$ opkg install libudev0
```

On some Linux distributions, the package `libudev1` will be available instead of `libudev0`. When compiling your application, you must link dynamically with both of these libraries, *libIBScanUltimate.so*, and several standard C++ libraries:

```
Build:$ gcc myapp.c -o myapp -l IBScanUltimate -l usb -l
udev -l stdc++ -l pthread
```

This command ignores the inclusion of the IBScanUltimate includes and library directories. It also disregards any aliasing necessary to reference libusb and libudev so simply. For example, the actual installed libudev library may be named *libudev.so.0*, and a link may be necessary:

```
Build:$ sudo ln -s /lib/arm-linux-gnueabihf/libudev.so.1
/usr/lib/arm-linux-gnueabihf/libudev.so
```

Depending on the file installed by the package manager, the same links may be needed on the target system to run an application gathering fingerprints with IBScanUltimate.

Compiling the Sample Application

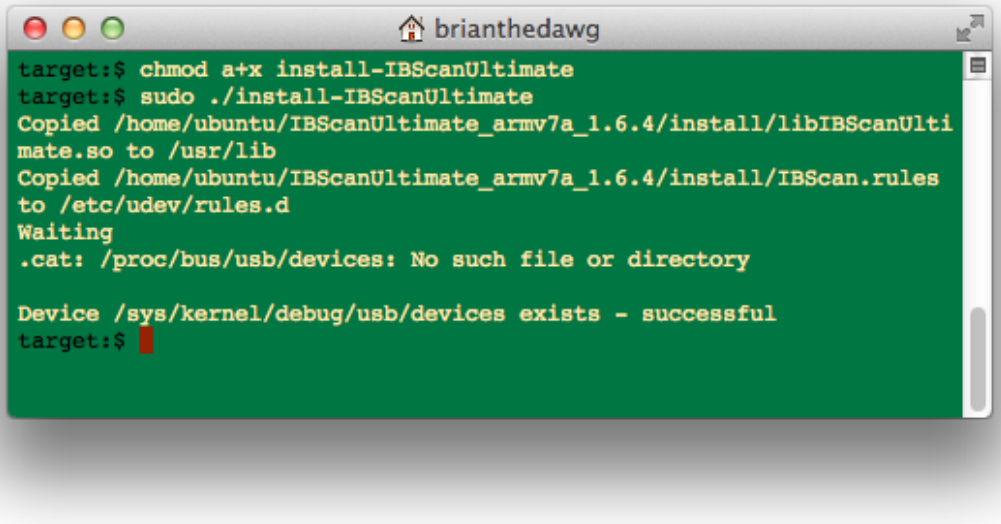
On some targets, the provided sample binaries may not execute because these are compiled against recent versions of the C runtime. To recompile the sample, navigate to the *Samples/Linux/TestScan* directory. Set and export the variable `CROSS_COMPILE` to the prefix of your tools, if you are not building natively, and set and export the variable `ARCHABI` to either `arm-linux-gnueabi` or `arm-linux-gnueabi`, depending on the ABI. Finally, make the application (*testScanU*), which will be output in a local *bin* directory.

```
Build:$ cd
IBScanUltimate_armv7a_x.x.x/Samples/Linux/TestScan
Build:$ export CROSS_COMPILE=/mygcc/bin/arm-linux-gnueabi-
Build:$ export ARCHABI=arm-linux-gnueabi
Build:$ make
```

If you are cross-compiling, native versions of libusb and libudev must be “installed” in your toolchain’s runtime library structure or on a library search path.

Installing the IBScanUltimate Library

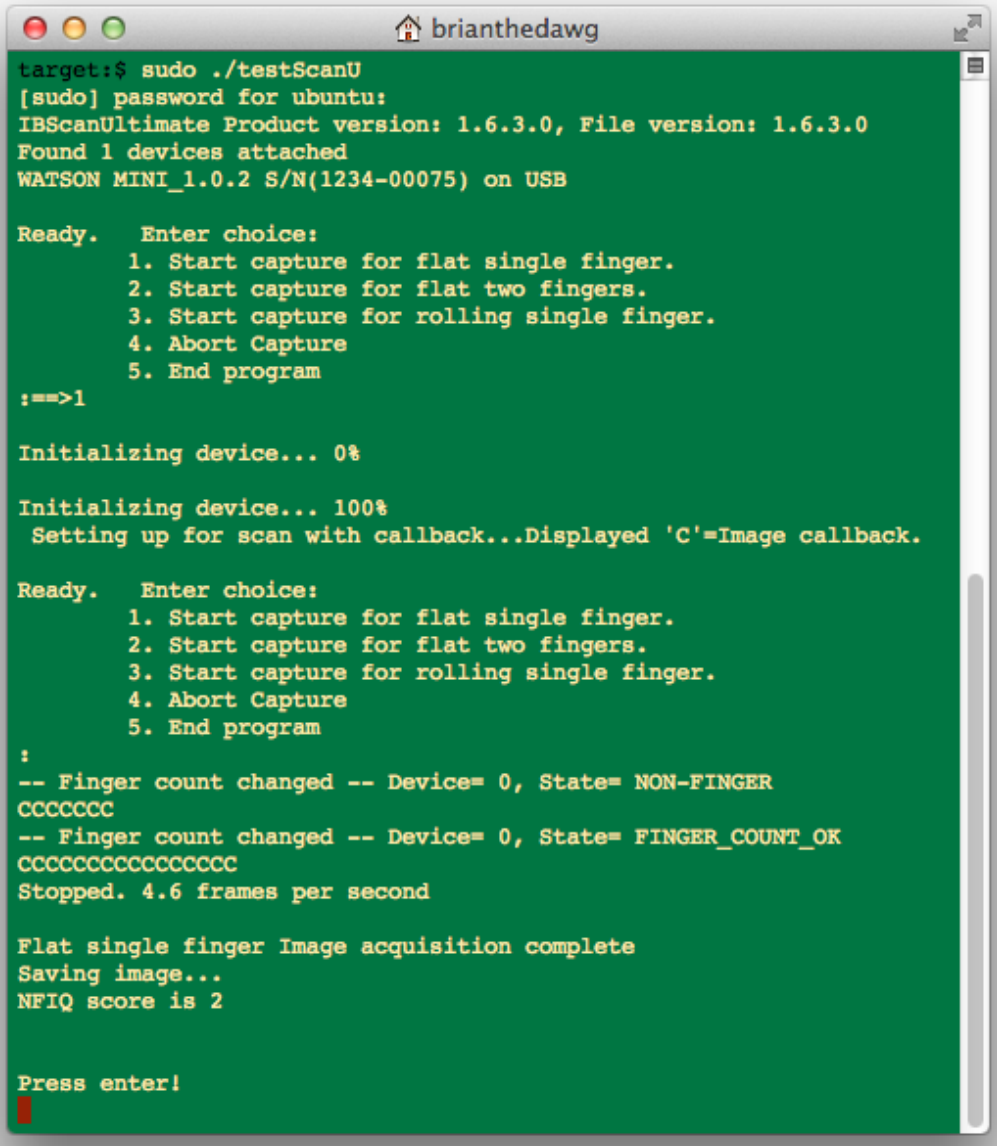
A script is provided to install the libIBScanUltimate library and configure udev for IB scanners. Transmit this script (*install-IBScanUltimate*) and accompanying rules file (*IBScan.rules*), found in the *install* directory of the distribution, to your target system, with the appropriate version of *libIBScanUltimate.so*, and (optionally) the compiled sample application.



Once the files are located on your target, plug an IB scanner into a USB port. Then

execute the script, requesting root access if necessary

```
Target:$ chmod a+x install-IBScanUltimate
Target:$ sudo ./install-IBScanUltimate
```



Run the sample application

Now you can run the sample application:

```
Target:$ sudo ./testScanU
```

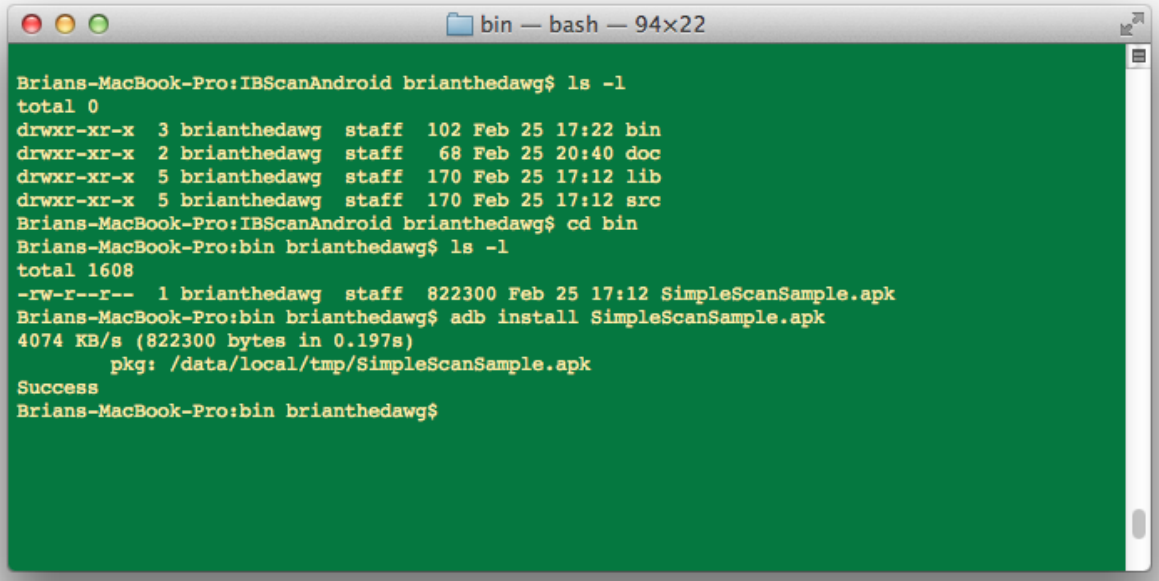
Depending on the user's permissions for accessing USB devices, you may need to grant root access to run the sample program.

2.4. Android Installation Guide

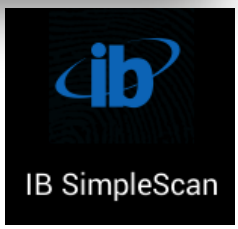
SDK Contents

The SDK contains the applications and libraries needed to start developing an Android application that interfaces with an IB scanner. The material is separated into five directories:

- The */bin* directory contains only one file: a compiled version of the IB SimpleScan application. To install this application, open a terminal window, move to that directory and use ADB to install the application on your connected Android phone or tablet. Make sure that *adb* (or *adb.exe*, on Windows), located in the Android SDK's *platform-tools* directory, is on the PATH or referenced by its full path.



If you have more that one Android device connected or have an emulator open, you may need to use the ADB –s switch with the appropriate identifier to direct the installation to your desired target. You will find the installed app listed under the name "IB SimpleScan".

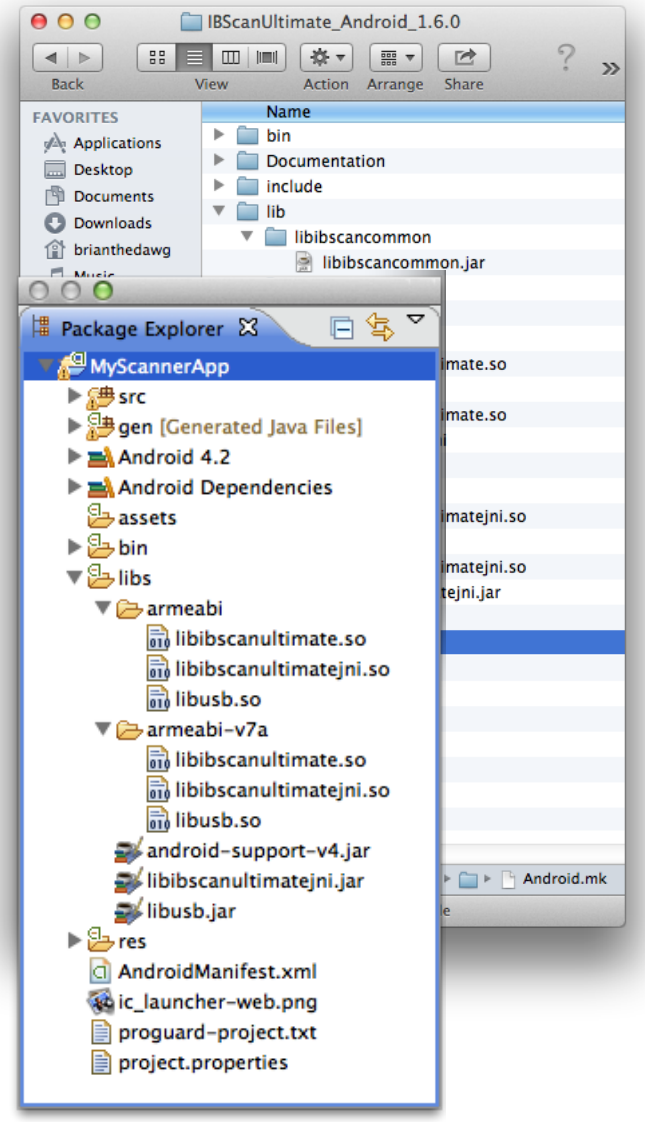


- The */lib* directory contains four folders, each containing compiled versions of the libraries you will need for development.
- The */include* directory contains the include files for the native interface of IBScanUltimate.
- The */Samples* directory contains source code for the SimpleScan sample application, the libusb distribution specifically for unrooted Android systems, and the wrapper which exposes to app a convenient Java interface for communicating with IB scanners.

Setting up an Eclipse Project with IBScanUltimate

As shown to the right, the */lib* folder contains, directly, four folders (*libibcommon*, *libibscanultimate*, *libibscanultimatejni*, and *libusb*) for the three native libraries and one pure Java library in the IBScanUltimate SDK; two native libraries have associated Java components (as evidenced by the JAR files in *libusb* and *libibscanultimatejni*). The *Android.mk* files assist in building other native libraries using the SDK's native libraries; since most apps should use the Java interface, these files can be ignored.

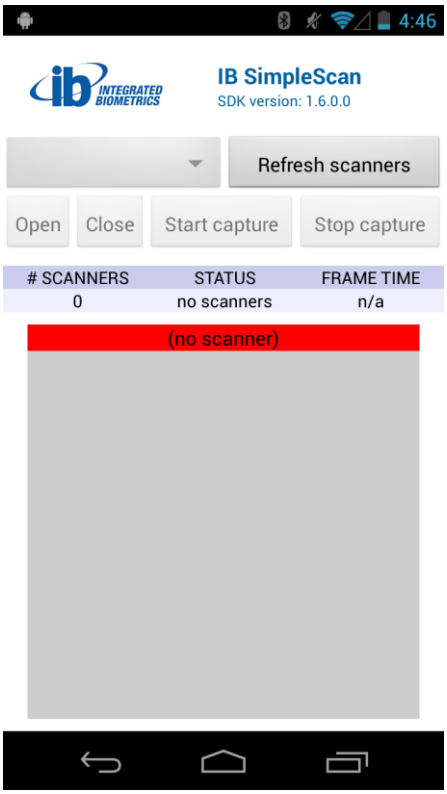
To set up an Eclipse project to use IBScanUltimate, copy the *armeabi* folders (one on top of the other on top of the other), *armeabi-v7a* folders (one on top of the other on top of the other), and JAR files into the *libs* directory; within the Project Explorer, the layout will appear as toward the right.



Opening and Running the Example Project

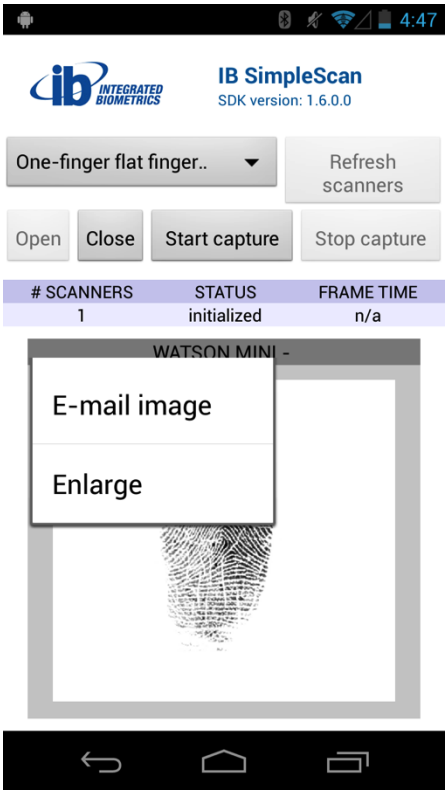
The IB SimpleScan example project is located in the */Samples/Android/SimpleScanSample* directory. To open this project in Eclipse

- Click “File” → “Import…”
- In the dialog box, choose “Existing Android Code Into Workspace” under “Android” and press “Next >”
- “Browse…” to the */src/SimpleScanSample* directory and optionally click “Copy projects into workspace”
- Press “Finish”



You should be able to run the sample on your Android device by right-clicking on the project title (“SimpleScanSample”) and choosing “Run As”→”Android Application”. Depending on your settings, a dialog may appear asking you to choose a connected Android device.

The sample will appear under the app list with the name “IB SimpleScan”. When opened, it will detect any attached scanners or wait until it received notification that a scanner has been attached. (On Android versions prior to 4.2.0, the device-attached broadcast does not fire when an app, like this one, declares no device filter in its *AndroidManifest.xml*; the “Refresh scanners” button is provided for those platforms.)



After a scanner is attached, select the scan type and press the “Open” button to initialize the scanner then the “Start capture” button to capture a scan. After the scan has complete, long-clicking on the image view with the fingerprint will reveal a pop-up menu with options to e-mail the image and show an enlarged version.

The IBScanUltimate Java Application Interface

The SDK libraries provide a Java interface for the convenience of the application, all within the package `com.integratedbiometrics.ibscanultimate`. The five principal classes of this package expose all of the functionality of the native IBScanUltimate libraries on Linux, with a more optimal interface for Java programming. For information beyond the brief introduction below, please see the Java API manual.

- `IBScan`: The scanner manager. The primary class, `IBScan`, manages scanners. Typically, an app's main Activity will get a handle to the single instance of the IBScan class in its `onCreate()` method:

```
this.m_ibScan =
    IBScan.getInstance(this.getApplicationContext());
```

  With this handle, the application can use the services IBScan provides, including getting a description the SDK version, getting the description of a device, and opening a handle to a scanner.

- `IBScanListener`: The scanner manager event listener. Typically, the application will register an IBScanListener to receive notifications:

```
this.m_ibScan.setScanListener(this);
```

  In this case, the Activity implements the IBScanListener interface itself, e.g.,

```
public class MyActivity extends Activity
    implements IBScanListener {
```

  The listener must override a number of methods. Several (covered more in the section on USB devices below) alert the app about events important to the cycle of device attachment (attached, permission granted, device count changed, detached). The other two are important to the procedure of opening a device.

- `IBScanDevice`: The scanner handle. A handle to a scanner is returned from one of the synchronous `IBScan openDevice()` functions, or through the `scanDeviceOpenComplete()` callback when using one of the asynchronous `openDeviceAsynch()` functions.

- `IBScanDeviceListener`: The scanner event listener. Typically, the application will register an IBScanDeviceListener to receive notifications:

```
this.m_ibScanDevice.setScanDeviceListener(this);
```

  In this case, the Activity implements the IBScanDeviceListener interface itself, e.g.,

```
public class MyActivity extends Activity implements
                                IBScanDeviceListener {
```

  The listener must override a number of methods. All of these are called during capture (after calling `beginCaptureImage()`) to allow the app to respond dynamically by, for example, updating its display of the current scan without needing to poll the device.

- IBScanException: The scanner exception. Most `IBScan` and `IBScanDevice`

returns may throw an `IBScanException`, which the app must catch. An exception may warn that an operation is illegal (such as setting a read-only property) or that an error occurred (such as a I/O error from a communication failure).

```
int deviceIndex = 0;
try
{
     IBScanDevice myDevice = this.m_ibScan.open(deviceIndex);
}
catch (IBScanException ibse)
{
     System.out.println("Failed to open with exception " +
         ibse.getCode().toString();
}
```

The IBScanUltimate C/C++ Application Interface

In general, the Java API will be the best choice for an application wishing to control an IB scanner, since it takes care of the messy details of wrapping the native IBScanUltimate libraries. Since some applications may prefer to code certain operations (for example, image processing) in C or C++, the native library interface is accessible. Conveniently, this Android C/C++ API is identical to the Linux API, and can be used in the same way (except for peculiarities in using USB devices on Android; see the next section). Please see the Linux API manual for more information on that API.

Using USB Devices on Android

Android-based systems are fairly locked-down; an app can access little beyond the boundaries of its process and resources. Permissions to access some features, such as external storage, are advertised by the app in its manifest, and the user is warned when installing the app. For USB devices, however, permission is granted on a device-by-device basis. An app can request permission to access a certain device programmatically (or become a default device by registering interest in certain categories by vendor and product ID or class), whereupon the user is prompted to approve or deny in a popup dialog. If the user approves, the app can access the device.

As part of this `IBScanListener` interface, several methods must be overridden; for the purposes of this discussion, two are most relevant. When a IB scan device is attached, the `scanDeviceAttached()` method will be called, offering the app an opportunity to request permission to access it:

```
@Override
```

```
public void scanDeviceAttached(int deviceId)
{
    /* Check whether we have permission to access this device.
    * Request permission so it will appear as an IB scanner. */
    boolean hasPermission =
        this.m_ibScan.hasPermission(deviceId);
    if (!hasPermission)
    {
       this.m_ibScan.requestPermission(deviceId);
    }
}
```

After the application responds to request, another listener method (`scanDevicePermisionGranted()`) will be invoked; if permission is granted, IBScanUltimate will refresh the list of devices it maintains internally then notify the application that the number of connected devices has changed with `scanDeviceCountChanged()` so the app will realize more scanners are available. (Of course, this method will also be invoked if a scanner is detached, for the opposite reason.)

```
@Override
public void scanDeviceCountChanged(int deviceCount)
{
    /* The number of recognized accessible scanners has changed.
     * Let's refresh the list of scanners we can choose from. */

   .
   .
   .
}
```

If the app enters a state when Android's `UsbManager` is not notifying it of device attachments, the app may need to manually iterate through the manager's list of devices and request permission to access any inaccessible scanners:

```
/* Make sure there are no USB devices attached that are IB
 * scanners for which permission has not been granted.  For any
 * that are found, request permission; we should receive a
 * callback when permission is granted or denied and then when
 * IBScan recognizes that new devices are connected. */
UsbManager manager = (UsbManager)this.getApplicationContext()
    .getSystemService(Context.USB_SERVICE);
HashMap<String, UsbDevice> deviceList = manager.getDeviceList();
Iterator<UsbDevice> deviceIterator =
```

```
        deviceList.values().iterator();
    while (deviceIterator.hasNext())
    {
        UsbDevice device      = deviceIterator.next();
        boolean   isScanDevice = IBScan.isScanDevice(device);

        if (isScanDevice)
        {
            boolean hasPermission = manager.hasPermission(device);
            if (!hasPermission)
                this.m_ibScan.requestPermission(device.getDeviceId());
        }
    }
```

3. Fingerprint Capture Guide

3.1. Flat Print Captures

The finger should be both properly aligned and maintain contact with the scanner's case. As shown in Figure 1a, on a Watson Mini, the finger should oppose the USB cable and be perpendicular to the side. From Figure 1b, you can see that the finger contacts the plastic case, as necessary on a Watson Mini, Columbo, and Sherlock sensors; on a Watson scanner, contact must be made with the metal strip across the scanner bottom.

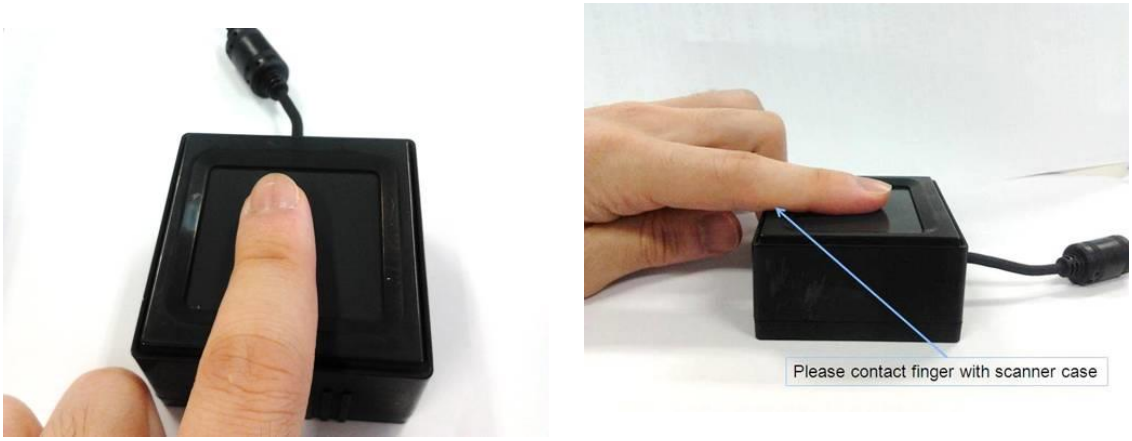


Figure 1. Good finger placement
(a, left) proper finger alignment; (b, right) finger contact with scanner case



Figure 2. Bad finger placement
(a, left) improper alignment, should be opposite USB cable; (b, middle) improper alignment, should be opposite USB cable; (c, right) no contact with scanner case

3.2. Rolled Print Captures

The recommendations for flat captures apply to flat captures, too. Both proper fingerprint alignment and contact with the scanner ground are necessary. The rolled capture is a sequence of three steps. As shown in Figure 3, the finger should be kept on the sensor surface until the flat print capture completes; in our sample programs, this condition is indicated with a red line (the line that tracks the capture location) in the fingerprint display.



Figure 3. Rolled capture, step 1. The finger should be kept on the sensor surface until the flat print capture completes

Then, as shown in Figure 4, the finger should be rolled left until the entire left side is captured; in our sample programs, this condition is indicated when the line that tracks the capture turns from red to green.

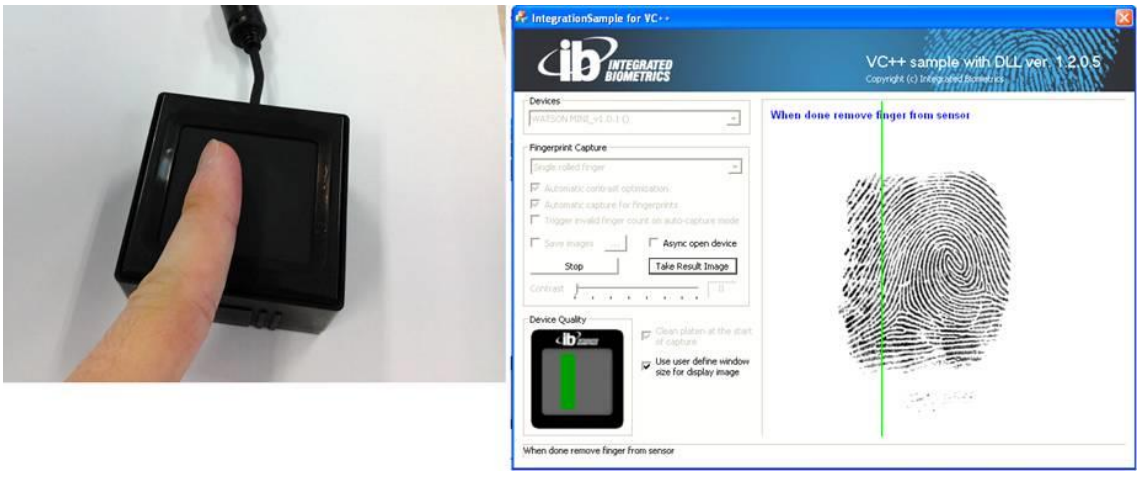


Figure 4. Rolled capture, step 2. The finger should be rolled left until the entire left side is captured

Finally, as shown in Figure 5, roll the finger back to the right until the entire right side is captured. The rolled capture completes when the finger is removed from the platen.



Figure 5. Rolled capture, step 3. The finger should be rolled right until the entire right side is captured

Support Contact Information:

www.integratedbiometrics.com

Integrated Biometrics, LLC

North American Office

Physical Address for Package Delivery
121 Broadcast Drive
Spartanburg SC 29303

For Mailings & Correspondence
PO Box 170938
Spartanburg, SC 29301

US & Canada
(864) 990-3711
Toll-free (888) 840-8034
Extension 1 – Company Directory
Extension 2 – Technical Support
Extension 3 – Sales Support
Extension 4 – Marketing
Extension 5 – Accounting
Extension 0 – Main Line

Sales & Pricing Inquiries
sales@integratedbiometrics.com

Terms & Conditions of a Sale

Terms & Conditions for Supplier Purchases

Sales Administration
marci.bowers@integratedbiometrics.com

Technical Support
technical@integratedbiometrics.com

South Korean Office

Physical Address and Mailing Address
#910 Suntech-City1, 513-15
Sangdaewon 1-dong Jungwon-gu
Seongnam-si, Gyeonggi-do
Republic of Korea

Phone
+82-31-777-2207

Sales Administration
everun@ibkr.co.kr