ANT™

the power of less

Creating ANT Android Applications

D00001576 Rev 4.2.0

Table of Contents

1 Overview

The purpose of this document is to provide developers the information (or the location of information) they need to create applications which run on the Android platform and utilize ANT wireless technology. It is intended to be used by developers looking to directly control ANT channels for non-ANT+ implementation.

More information on ANT, ANT+, and related downloads and documentation, visit the ANT Wireless website http://www.thisisant.com.

1.1 SDK

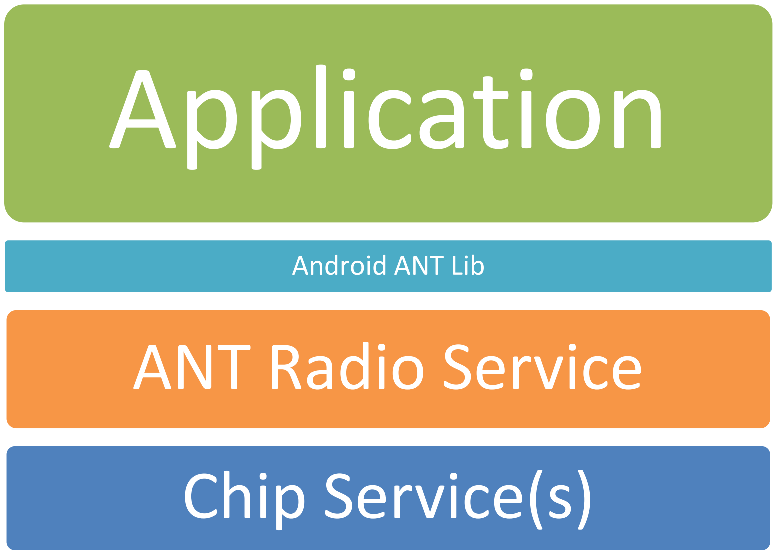
The Android ANT SDK Package contains everything you need to develop ANT applications for Android. It includes the Android ANT Lib (jar file and javadoc), prebuilt sample applications and sample code, the ANT Radio Service and the ANT USB Service (both available via the Google Play Store).

1.2 ANT and ANT+

For an overview of the difference between ANT and ANT+, please see http://www.thisisant.com/developer/ant-plus/ant-antplus-defined/

2 ANT Application Requirements

Before developing Android ANT applications, there are a few ANT specific dependencies that need to be addressed. To develop or run applications using ANT Wireless technology an ANT chip, its associated chip service (e.g. a built-in service, ANT USB service, or ANT emulator bridge), and the ANT Radio Service is required. Developers require the Android ANT Library to access the provided ANT functionality.



2.1 Chip Services

In order to utilize ANT in Android, ANT support is required on the target device in the form of the ANT Radio Service and a chip service. These can be present as a built-in feature of the Android device, via an external accessory and service such as an ANT USB stick and ANT USB service (for devices with the USB-Host feature), or through the ANT Emulator Bridge to connect to ANT USB stick on a PC.

2.1.1 Built-in ANT Chip

A list of officially supported devices with built-in ANT support is maintained in the ANT Product Directory under the Mobile Phones/Devices category. The ANT Radio Service and chip service are built-in on devices with built-in ANT support. When using external ANT accessories in addition to a built-in chip both the built-in's and the accessory's chip service must be installed.

2.1.2 ANT USB

The ANT USB Service is required to communicate with ANT USB sticks using the Android USB Host feature. ANT USB Service is an application that can be installed and updated via the Google Play Store, and is also included in the SDK. Note: not all Android devices support the USB Host feature. Also, Android applications can use ANTUSB-m, ANTUSB2, Suunto Movestick Mini and newer USB sticks. ANTUSB1 is not supported.

2.1.3 Third-party Add-ons

It is possible to use other add-on ANT chip providers besides the built-in chip and ANT USB. These work in much the same way as the ANT USB, but using custom hardware instead of an ANT USB Stick. Chip services for these third-party add-ons must be installed. See the chip provider for details.

2.1.4 Emulator Bridge

For ANT support on the Android Emulator, an Emulator Bridge has been made available for use by developers. See section 7.2 below.

2.2 ANT Radio Service

The ANT Radio Service manages controlling the ANT radio (or radios) and sharing it between multiple applications. It is required to utilize ANT on an Android device. This is an application that can be installed and updated via the Google Play Store, and is also included in the SDK.

2.3 Android ANT Lib

The Android ANT Lib library defines the ANT API that allows an application to acquire the ANT channels it requires and configure these channels for wireless communication. Developers can add this library to their applications to be able to utilize the full set of ANT functionality made available through the ANT Radio Service. The Javadoc for the API is available online at http://www.thisisant.com/APIassets/Android_ANT_API/index.html

2.4 ANT+ Membership and License Agreement

In addition to the services and libraries, in order to receive support directly or through the ANT developer forums, you must sign up for an account on http://www.thisisant.com. Signing up for an 'adopter' level account is free and provides access to most resources. Details on the various levels of membership are explained on the website.

3 Using the ANT API

3.1 Overview

An application can communicate using ANT by following these general steps:

1) Check for ANT support
2) Bind to the ANT Radio Service
3) Get the ANT Channel Provider
4) Acquire Channel(s)
5) Configure Channel(s)
6) Use Channel(s)
7) Release the Channel(s)
8) Unbind from the ANT Radio Service

3.1.1 Checking for ANT Support

In order for applications to access and use the ANT protocol, developers should first check if ANT support exists. See section 2.1 for details on the different types of ANT support. The AntSupportChecker class in the Android ANT API provides static helper methods for determining if the application is running on a device with an ANT adapter.

3.1.2 Built-in ANT

Built-in ANT support can be checked using hasAntFeature() method. This method checks if the application is running on a device with ANT support provided by a built-in chip provided by the phone vendor.

3.1.3 External ANT Accessory

Additional services must be installed before an application is able to access external ANT accessories (see section 2.1). The hasAntAddOn() method will check for the presence of any of these ANT chip provider services. This does not necessarily mean that the required hardware is attached, just that the user has installed the service.

3.1.4 Installing Required Services

There are cases where an application is running on a device which supports (or could support) ANT, however the user must install the required services. It is possible for an application to determine if this is the case.

3.1.4.1 ANT Radio Service

There is no explicit check to see if the ANT Radio Service is installed, however if a bind request fails the user should be asked to install ANT Radio Service, available on the Google Play Store.

3.1.4.2 ANT USB Service

The ANT USB Service can run on Android devices with the USB Host feature.

3.2 Binding to the ANT Radio Service

To use the ANT chip, whether it is built-in or an add-on, applications must bind to the ANT Radio Service. Applications can call the static method bindService() within the AntService class to bind to the ANT Radio Service. This will return true if the service was successfully bound. If it returns false, a connection could not be made and the ANT Radio Service may not be installed.

When binding to the service, the method receives an Android API ServiceConnection object which will implement onServiceConnected() and onServiceDisconnected() methods. Like any other Android service, these methods will be called when the service is connected and disconnected and provide an Android IBinder object which is the interface that can be used to communicate with the bound service.

For the Android ANT API, the IBinder instance received within the onServiceConnected() method must be passed into the AntService constructor. The application will then be able to use the methods within AntService and communicate with the ANT Radio Service.

3.3 Acquiring an ANT Channel

3.3.1 Channel Provider

The AntService class provides the getChannelProvider() method which returns an AntChannelProvider object and is the interface that applications can use to acquire channels from the ANT Radio Service. AntChannel objects are returned when a channel is acquired from the provider after which the application can configure and use the acquired channel with the methods provided.

AntChannelProvider also provides methods that can be used to check how many channels are available or how many channels are available with a given set of capabilities.

3.3.2 Capabilities

The AntChannelProvider allows applications to either acquire any available channel, or a channel with specific capabilities. For information about specific capabilities, see section 5.

3.3.2.1 Required Capabilities

Any capability set in the required capabilities will ensure only a channel with those capabilities is acquired.

3.3.2.2 Desired Capabilities

Desired capabilities are not guaranteed to be available in the acquired channel, but if multiple ANT adapters are available, the ANT Radio Service will try its best to select a channel from the adapter which best matches these capabilities.

3.3.3 Network

When acquiring a channel, applications must specify the specific ANT network that the channel being acquired will participate in.

3.3.3.1 Predefined Networks

Public, ANT+, and ANT-FS are PredefinedNetworks where a network key is not required. The Public network has no restrictions in regards to its use, but the ANT+ and ANT-FS network define rules and specific data formats to ensure device interoperability for all devices participating in these networks. When participating in the ANT+ and ANT-FS network, the application must adhere to the rules and behaviors defined by these networks. For more information about ANT Networks please see section 5.2.5 of the ANT Message Protocol and Usage Document.

The ANT+ network is not accessible through the standard ANT API, and it is intended that developers creating applications which talk to ANT+ devices use the ANT+ API. For special cases requiring the ANT+ network, please contact info@thisisant.com.

3.3.3.2 Private Networks

This functionality is not currently available in the Android ANT API.

3.3.4 Wait for a Channel

If there are no channels available, acquireChannel() in AntChannelProvider will throw a ChannelNotAvailableException which will contain the ChannelNotAvailableReason why no channels are available. Applications should implement a Broadcast Receiver that checks for Intents that match ACTION_CHANNEL_PROVIDER_STATE_CHANGED in AntChannelProvider to receive information about the availability of channels. This intent contains extras that report if new channels are available, the number of channels available and if the legacy interface is in use (i.e. an application claims and uses the entire ANT adapter) all retrievable by the keys provided within AntChannelProvider. If the legacy interface is in use, applications can attempt to acquire a channel with acquireChannel() which will then prompt the user to free the ANT radio. After the adapter and radio is freed, the application can successfully acquire a channel. Applications can determine how many channels are available by calling getNumChannelsAvailable() and if the legacy interface is in use by calling isLegacyInterfaceInUse().

3.4 Configuring and Using an ANT Channel

After acquiring an AntChannel instance, applications must establish a channel by first assigning, then configuring and opening the channel using the methods provided. For more details about establishing a channel and the information required to do so, please see section 5.3 of the ANT Message Protocol and Usage Document.

When calling assign(), the channel type must be specified with channel types being defined within the ChannelType enumeration and, optionally, the required extended assignment features (see section 3.4.5). Before setting the RF frequency, applications should check if the frequency falls within the valid range for the channel; for more information about and how to check the valid frequency range of the channel see section 5.5. See section 6 for more information on adapter wide configurations such as Lib Config and power saving settings. After configuration, the channel is now able to be opened. Once the channel is opened the application can set the broadcast, burst transfer or acknowledge message to be sent.

3.4.1 Channel Events

Channel Event handlers respond to received message events, in which the received message can be constructed, as well as channel death events indicating that the channel has become unusable. In order to receive channel events, applications must set and implement an IAntChannelEventHandler.

3.4.2 Adapter Events

Adapter Event handlers respond to Lib Config, power saving setting, burst transfer state, and background scan state changes. The adapter wide events must be handled independently for each channel as there is no guarantee that different channels exist on the same adapter. In order to receive adapter events, applications must set and implement an IAntAdapterEventHandler.

3.4.3 Parsing ANT Messages

There is no need for applications to implement their own parsing of the raw ANT packets. Each ANT message has a corresponding class in the *com.dsi.ant.message.fromant* package which will perform the parsing. The onReceiveMessage() method within the channel event handler contains a AntMessageParcel; applications can then use static method createAntMessage() in AntMessageFromAnt to construct and cast to an instance of the correct message type. After constructing an instance of the correct message type all information from the message can be simply requested through the provided methods.

3.4.4 Data Messages

3.4.4.1 Payload

Data payload is completely application dependent. A custom ANT implementation may be created using any data format (on the ANT+ network the ANT+ profiles define a standard data payload for ANT+ devices). The MessageUtils class provides helper methods for reading and writing information to the raw data payload.

3.4.4.2 Extended Data

Extended data is enabled through the LibConfig command (see section 6.4 for information about Lib Config). Once extended data is enabled, each Data Message will contain extra information, available with hasExtendedData() and getExtendedData().

3.4.5 *Extended Assignment Features*

After acquiring a channel capable of the desired extended assignment features, to enable these features on a channel applications must pass an ExtendedAssignment instance into assign() with the required features set to true.

3.4.5.1 Frequency Agility

Before configuring, a check for if the frequencies fall within the valid range for the channel should be performed. For more information about and how to check the valid frequency range of the channel, see section 5.5. To set the three operating frequencies to be used by the frequency agility feature, applications can use the configureFrequencyAgility() method in AntChannel.

3.5 Release the Channel(s)

Releasing a channel is necessary for making the channel available for use by other application. To release a channel, use the release() method in AntChannel. After a channel has been released, the instance can no longer be used. Any attempt to use the channel will result in a RemoteException. Channels may also be release because of external causes, such as a USB Stick being removed or the user giving permission to an application using the legacy interface to take over all channels. After a channel is released, regardless of what caused its death, an onChannelDeath() occurs.

3.6 Unbind from the ANT Radio Service

To unbind from ANT Radio Service, first unregister any receivers linked to the ANT Radio Service, such as the AntChannelProvider state changed broadcast receiver, and then unbind from ANT Radio Service as you would when unbinding from any Android Service.

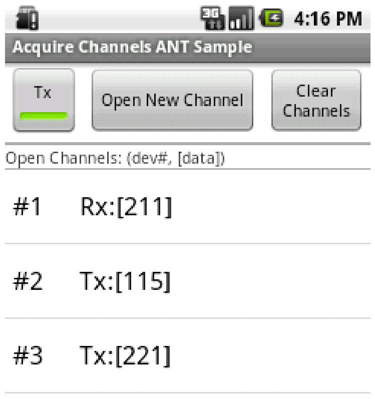
Note that the ANT Radio Service only runs as a bound service, meaning that if your application is the only one left bound, on unbind, the ANT Radio Service will shutdown. This means that when your application re-binds, ANT Radio Service will re-initialize. Your application will have to wait for the SERVICE_INITIALISING flag to clear before channels are available once again.

4 Sample Applications

To complement the Javadoc included in the SDK, these reference applications (provided as Eclipse projects) show how to use the ANT API.

4.1 Acquire Channels

Shows how to acquire ANT channels and configure them for simple communication with other devices. Opening an Rx channel on one device will find and receive data from the matching channel opened as a Tx channel on another device. This is the most basic communication, where a master channel broadcasts data and any slave channel connected receives that data.



Open Channels: (dev#, [data])

#1 Rx:[211]

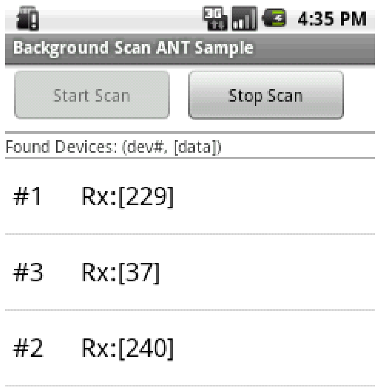
#2 Tx:[115]

#3 Tx:[221]

When the app starts, it will bind to the ANT Radio Service. The “Open New Channel” button will become available when ANT Radio Service is connected, and stay available while there are free ANT channels. Note that channels will be acquired from all available ANT adapters. For example, with a built-in ANT adapter and an ANT USB stick connected (with the ANT USB Service), 16 independent channels may be used.

4.2 Background Scan Sample

Shows how to acquire, configure and open a background scanning ANT channel. When a channel that is capable of background scanning is acquired (by passing in a Capabilities object with support for background scanning set to true) and another background scan is not in progress, the “Start Scan” button become available. This button will become unavailable if background scan capable channel is not acquired or a background scan is already in progress. Starting a background scan will find Tx channels open on other devices and display the most recently received data to the screen as well as the device number. Getting the device number is achieved by setting the Lib Config which configures which extended data (such as channel ID) will be included in received messages. The scan will run continuously with no timeout and will only be stopped when explicitly told to stop by pressing the “Stop Scan” button.



Found Devices: (dev#, [data])

#1 Rx:[229]

#3 Rx:[37]

#2 Rx:[240]

5 Capabilities

The capabilities of a channel in the Android API is a subset of the possible “ANT Product Capabilities” (features that are supported on specific hardware; see section 9.4 of the ANT Message Usage and Protocol Document), limited to those which are meaningful on the ANT hardware Android applications will encounter; developers only need to be concerned with the capabilities listed in Capabilities. Some ANT hardware does not support all of these capabilities. As such, it is important to check for required or desired capabilities when acquiring a channel (see section 3.3.2). The Capabilities object is used when acquiring channels and when querying a channel for what operations it supports. This section describes the different ANT capabilities available to Android developers.

5.1 Extended Data

Data type messages can be extended to allow ANT to pass additional information to the host, along with the received data message. See section 3.4.4.2 for more information about extended data and how to enable it.

5.1.1 Received Message Timestamps

The channel is capable of receiving messages that include a timestamp (as a counter) of when the message was received.

5.1.2 RSSI

The received signal strength indication (RSSI) capability allows a channel to receive data about the RSSI value of incoming messages. This data can then be used to determine proximity between the device your application is operating on and other transmitting devices. RSSI can also be used instead of the proximity search feature to allow more flexibility in acquiring the appropriate device.

5.2 Extended assignment

Using extended assignment allows for various advanced ANT features to be enabled on that channel. See section 3.4.5 for information on how to enable extended assignment features when assigning a channel.

5.2.1 Background Scanning

A background scanning channel is a special channel type that operates in search mode; however, instead of acquiring a master, ANT will pass the data to the host and continue searching. This will perform a continuous low priority search until the channel is closed. See section 6.2 and the Channel Search and Background Scanning Channel application note for more information on background scanning.

5.2.2 Frequency Agility

Frequency agility allows a channel to change its operating frequency in the case of significant performance degradation to improve coexistence with other wireless devices such as Wi-Fi. This is especially useful in RF crowded environments. Both the master and slave need to have frequency agility enabled as well as have the same three operating frequencies configured. For more information please see the Frequency Agility application note.

5.3 Wildcard ID List

Wildcard ID List capability allows applications to set channel ID to the wildcard value of zero in the inclusion/exclusion list. This is useful when the application does not know the exact channel ID of the master it wants or does not want to connect to, and only knows a subset of the channel ID information. On adapters which do not support a wildcard, using one will result in the ID list never matching a device.

5.4 Event Buffering

The host MCU can use Event Buffering to limit the frequency at which low priority events are sent from the ANT device to the host MCU. By deferring the processing of events, the host MCU can remain in a low power state for a longer period of time and reduce its power consumption. See section **Error! Reference source not found.** for how to configure the event buffer power saving setting.

5.5 Frequency Range

Certain channels may only support a limited frequency range as compared to the range defined in section 5.2.2 of the ANT Message Usage and Protocol Document. To check the valid frequency range of the channel, getCapabilities() should be called to get the channel's Capabilities object which can then be queried for the minimum and maximum frequency values with the methods provided.

6 Adapter-Wide Configuration

The API allows for multiple applications to run concurrently on their own channel, without need for consideration of another channels in most cases. There are certain configurations which apply adapter-wide with each channel getting a vote for what they would like the configuration to be. For a more technical details on why this is the case, consult the Multi-Channel Design Considerations ANT Application Note.

After any channel triggers a change in an adapter-wide configuration, a state changed event is sent to all channels on that adapter. To receive these events applications must implement an IAntAdapterEventHandler and the respective methods corresponding to each event. An application must set an adapter event handler and handle adapter-wide events independently for each channel as there is no guarantee that all channels acquired by an application will exist on the same adapter.

Adapter-wide settings such as Lib Config, are set by the channels on their respective adapter. The response to these request messages will be received by all channels on the adapter. This also applies when requesting a response for an adapter-wide value, such as ANT Version, Capabilities and Serial Number.

6.1 Burst

ANT only allows for one burst transfer to occur on an adapter at a time, and a failure with TRANSFER_IN_PROGRESS reason will occur if a second channel on an adapter attempts to perform a burst transfer while one is in progress. When onBurstStateChange() indicates that the BurstState is not processing, then it is safe to attempt the burst on the second channel.

To prevent burst transfers from drastically impacting other open channels, the API limits burst transfers to a maximum size. Attempts for burst transfers that exceed the maximum size will fail with an INVALID_REQUEST reason. Maximum burst size may change depending on the current usage of the ANT adapter. Changes to the maximum burst size will trigger an onBurstStateChange() event. Applications should check the new Burst State parameter received by onBurstStateChange() for the new maximum burst size. The current maximum burst size is also accessible by getting a Burst State instance from the methods in Ant Channel. For more information please see the Burst Transfers application note.

6.2 Background Scan

On a single adapter, multiple channels can be configured to be background scanning channels. That being said, only one background scan is allowed to occur at a time on a single adapter. A failure, with BACKGROUND_SCAN_IN_USE reason, will occur when attempting to open a second background scan channel. When onBackgroundScanStateChange() indicates that the BackgroundScanState is not in progress, then it is safe to open the next background scanning channel. Background scan channels should be used in conjunction with extended data configured by a Lib Config so that when the channel receives messages the application can retrieve useful information such as RSSI data, timestamp and channel ID of the device found.

Background scan channels run forever (i.e. no search timeout will occur) and will only close if the channel is explicitly closed by the application. Thus, setting timeout values for background scan channels will result in an 'Invalid Request' command failure reason and should not be set. Background scan channels will only operate in low priority search mode.

6.3 Continuous Scan

Continuous scan is not possible in the Android API as this feature prevents other channels from transmitting or receiving messages. To emulate the behavior of a continuous scan, an application can use a single background scanning channel and one or more slave channels. For more information of Continuous scans please see the Continuous Scanning Mode application note.

6.4 Lib Config

Lib Config messages can be sent to the ANT adapter to set which extended data (Timestamp, RSSI) will be passed to the Host along with the received data message (see section 9.5.19 of the ANT Message Protocol and Usage Document). The API accounts for multiple channels attempting to set Lib Config for the adapter: When a channel sets a Lib Config, the channel's desired settings are added to an adapter-wide list of desired settings. A Lib Config message is then sent to the adapter with the newly desired settings, along with other desired settings expected by other channels. The response to the Lib Config message will be sent to all channels on the adapter.

Applications can send a Lib Config message using setAdapterWideLibConfig(), passing in a LibConfig containing the desired configurations. A Lib Config message will trigger an adapter-wide onLibConfigChange() event which applications can receive by implementing the respective method in the adapter event handler.

6.5 Power Saving

Power saving settings can be set by channels for additional and significant power savings. Applications may request specific power saving settings to save power if desired such as when the screen is off or when the application is not required to send real time data. Currently, only the event buffer power saving feature is included in the API. Since these features have adapter-wide affects, channels are not guaranteed to have their requested power saving settings applied as the adapter-wide settings as the interests of all channels need to be considered. For this reason, the power saving settings that will be applied will be the minimum power saving settings requested.

Event buffering is a power saving setting where the ANT device will limit the frequency at which low priority events are sent to the application. By setting the maximum time events will be buffered for, applications are guaranteed that buffered events will be flushed before or when this maximum time is reached. The buffer may be flushed before this set maximum time if a high priority event occurs or if another channel sets the maximum buffer time to lower than the maximum time set previously. The adapter-wide event buffer maximum time is set by determining the lowest maximum time for the buffer set by all channels currently open on the adapter in order to fulfill the guarantee to all channels that events will be flushed before or when their maximum buffer time is reached. By default, channels set the maximum buffer time to 2 seconds, the optimal time for balancing responsiveness and power savings. Setting the maximum buffer time lower than the default will sacrifice the power savings provided by event buffering. For more information on the event buffer power saving feature, see section 9.5.2.22 of the ANT Message Usage and Protocol Document.

Before setting any power saving settings, applications should acquire and determine if the channel is capable of the desired power saving settings via Capabilities (see section 5). Applications can request maximum buffering time by creating an EventBufferSettings object with the desired maximum buffering time and then passing it into a call of setEventBuffer(). Changes to the event buffer settings of the adapter will trigger an onEventBufferSettingsChange() and must be implemented if applications want to receive event buffer change events. Applications can get the current EventBufferSettings via getEventBufferSettings().

6.6 Other Advanced Features

The Android ANT API currently does not support advanced configuration features such as encryption, event filtering or selective data updates.

7 Development Tools

7.1 Android SDK

All Android application development requires the Android SDK. The required files and the information to get started at http://developer.android.com/sdk/index.html

7.2 ANT Android Emulator Bridge

This ANT Android Emulator Bridge tool can be used with Android Virtual Devices (AVD's). The bridge allows an ANT USB stick that is connected to the PC running the bridge to be used by the AVD over the PC's network interface. ANT Android applications can then be developed without the use of a physical Android device. The Emulator Bridge tool is available for download in the ANT Developer's Zone at:

http://www.thisisant.com/developer/resources/downloads/

The 'ANT Android Emulator Bridge Tool' downloadable package contains the Bridge Tool application along with documentation explaining how to use the tool, as well as the required Emulator Configuration Android application.

8 Reference and Support

8.1 ANT Message Protocol and Usage Document

The Android ANT API allows Android applications to control the ANT radio, interacting via the ANT Message Protocol. While there are some differences to the serial interface described therein, most of the API methods correspond to messages described in section 9.5 (ANT Message Details) of the ANT Message Protocol and Usage Document. This can be found in the ANT Developer's Zone at

http://www.thisisant.com/resources/ant-message-protocol-and-usage/

8.2 Developer Forums

The ANT Developer's Forums provide a place to search for information, ask questions regarding ANT development, and find details on how to handle common issues you may encounter.

http://www.thisisant.com/forum/

8.3 Android Developers Guide

Android application development has several key differences when compared with developing for other operating systems. Knowledge of the Android framework and the Android application lifecycle will be essential to creating successful ANT Android applications. A good starting place is the Android "Getting Started" guide:

http://developer.android.com/training/index.html

9 Notes on Use of ANT+ Logos and Certification

The ANT+ logos are used to inform consumers of an application's interoperability with ANT+ devices. Only certified applications are allowed to use the ANT+ name, logos or icons.

NOTE: Applications seeking ANT+ certification should be using the ANT+ API, not implementing the profiles themselves on the ANT API.

Before using any ANT+ branding, the application must complete the ANT+ certification process to ensure that it complies with the device profiles it implements. The process is similar to the certification process for ANT+ sensors and devices but is streamlined for applications. This process is further streamlined for applications that make use of the ANT+ Plugins. For more details please visit: http://www.thisisant.com/developer/ant-plus/certification/ or contact certification@thisisant.com.

Once certification is complete the ANT+ logos can be used on both the application and promotional materials such as on the Google Play Store or other websites where the application is available. Logo files will be distributed upon certification.