ANT™
the power of less

ANT Message Protocol and Usage

Copyright Information and Usage Notice

Revision History

Revision	Effective Date	Description
4.3	June 2011	Updated formatting, Added section 8.2.1, Added extended messaging
4.4	October 2011	Added Channel Search Priority, Added section 9.4.3
4.5	November 2011	Changes to channel status byte
5.0	January 2013	Revisions to add new nRF51 and ANTUSB-m feature information: - Advanced Burst Transfer - Single Channel Encryption - Event Buffering - Event Filtering - Fast Channel Initiation - Asynchronous Transmission - High Duty Search - Selective Data Updates Condensed section 2 The ANT Product Family, and updated the document template.

Table of Contents

List of Figures

List of Tables

List of Equations

1 Introduction

ANT™ is a practical wireless sensor network protocol running in the 2.4 GHz ISM band. Designed for ultra-low power, ease of use, efficiency and scalability, ANT easily handles peer-to-peer, star, connected star, tree and fixed mesh topologies. ANT provides reliable data communications, flexible and adaptive network operation and cross-talk immunity. ANT protocol stack is extremely compact, requiring minimal microcontroller resources and considerably reduces system costs.

ANT provides carefree handling of the Physical, Network and Transport OSI layers. In addition, it incorporates key low-level security features that form the foundation for user-defined sophisticated network security implementations. ANT ensures adequate user control while considerably lightening computational burden in providing a simple yet effective wireless networking solution.

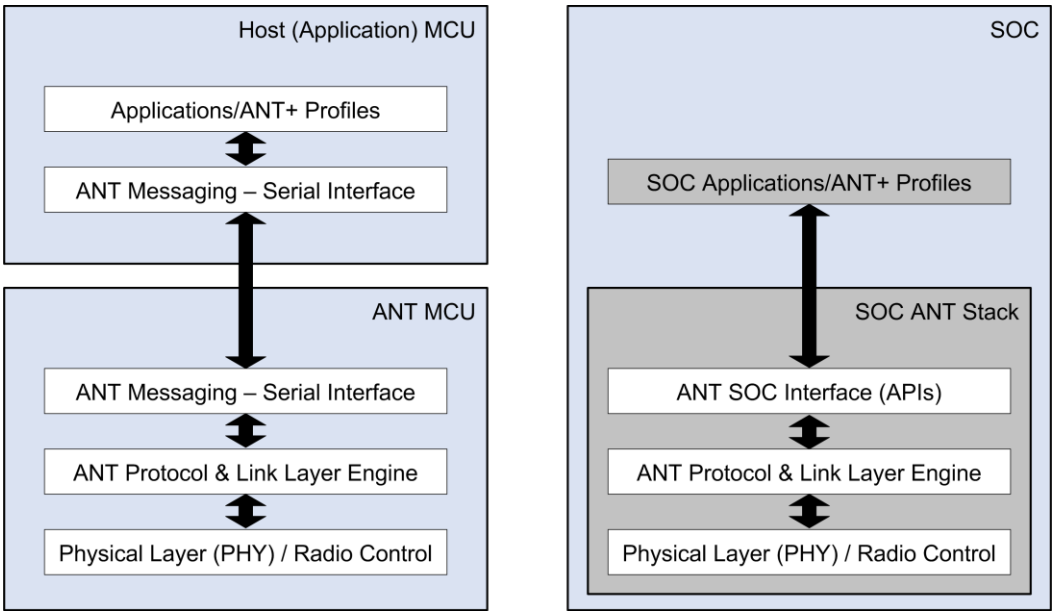


Figure 1-1. ANT Layers in Standard ANT/HOST and System On Chip Devices

The interface between ANT and the Host application has been designed with the utmost simplicity in mind such that ANT can be easily and quickly implemented into new devices and applications. The encapsulation of the wireless protocol complexity within the ANT chipset vastly reduces the burden on the application host controller, allowing a low-cost 4-bit or 8-bit Microcontroller (MCU) to establish and maintain complex wireless networks. Data transfers can be scheduled in a deterministic or ad-hoc fashion. A burst mode allows for the efficient transfer of large amounts of stored data to and from a PC or other computing device.

A typical ANT-enabled device consists of an application host MCU interfaced with an ANT module, chipset or chip. The host MCU establishes and maintains a communication session to other remote ANT-enabled devices by means of a simple, bidirectional, serial message protocol. This document details the protocol and provides examples of how to use ANT for wireless networking.

2 The ANT Product Family

ANT technology has been incorporated into a family of products that allows a particular implementation to be scaled to suit the needs of the application and the vision of the product designer. Details of the available ANT chips, chipsets, modules, USB sticks etc. are available online at www.thisisant.com/developer/components. In addition, a range of software tools, and comprehensive documentation have been provided for developers (www.thisisant.com/developer). Technical support is available via the ANT forum.

3 Network Topologies

The ANT protocol has been designed from the ground up to support a large range of scalable network topologies. It can be as simple as a 2-node unidirectional connection between a transmitting peripheral device and a receiver, or as complex as a multi-transceiver system with full point-to-multipoint communication capabilities.

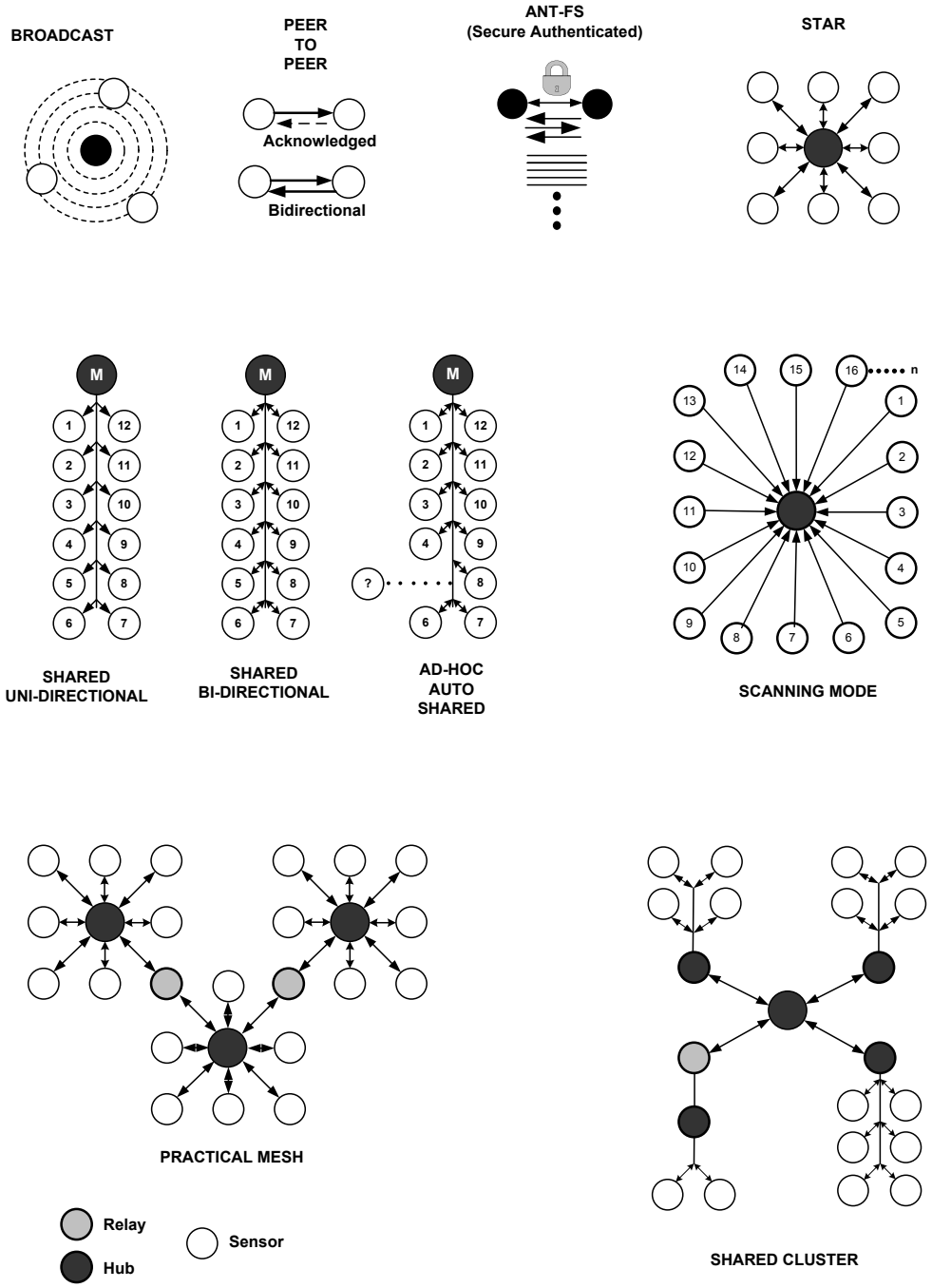


Figure 3-1. Example ANT networks

For the purpose of illustration, a simple example follows to demonstrate the basic concept of ANT channels (Figure 3-2).

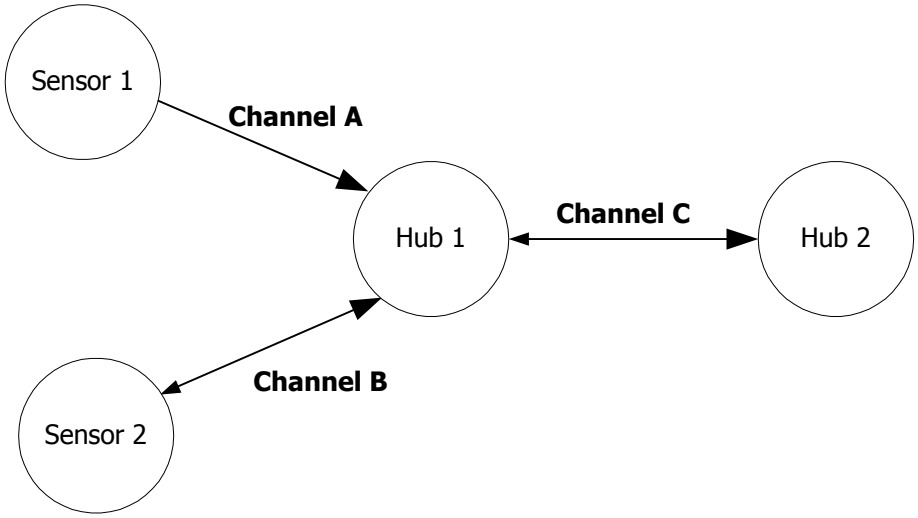


Figure 3-2. A simple ANT network

ANT usage and configuration is channel-based. Each ANT node (represented by a circle) can connect to other ANT nodes via dedicated channels. Each channel generally connects two nodes together; however a single channel can connect multiple nodes.

Each channel has, as a minimum, a single master and single slave participant. The master acts as the primary transmitter, and the slave acts as the primary receiver. The large arrows in Figure 3-2 indicate the primary data flow from master to slave; small arrows indicate reverse message flow (e.g. Channel B, C). A channel with a single arrow (e.g. Channel A) is used to represent a one-way link, which supports the use of lower-power transmit-only nodes. Note that an ANT node can act as both a slave (e.g. Hub1 channel A, B) and a master (e.g. Hub1 channel C) simultaneously.

Table 3-1 describes the master / slave status of each of the channels shown in Figure 3-2.

Table 3-1. Master/slave status of Figure 3-2 channels

Channel	Master	Slave
Channel A	Sensor1 (TX-Only)	Hub1 (RX)
Channel B	Sensor2 (TX)	Hub1 (RX)
Channel C	Hub1 (TX)	Hub2 (RX)

4 ANT Nodes

Each node in an ANT network consists of an ANT protocol engine and a host controller (MCU). The ANT engine encapsulates the complexity of establishing and maintaining ANT connections and channel operation within its firmware. The host controller is thus free to handle the particulars of an application with only a limited burden in initiating ANT communications to other nodes, which it does via a simple serial interface between host and ANT engine, as shown in Figure 4-1.

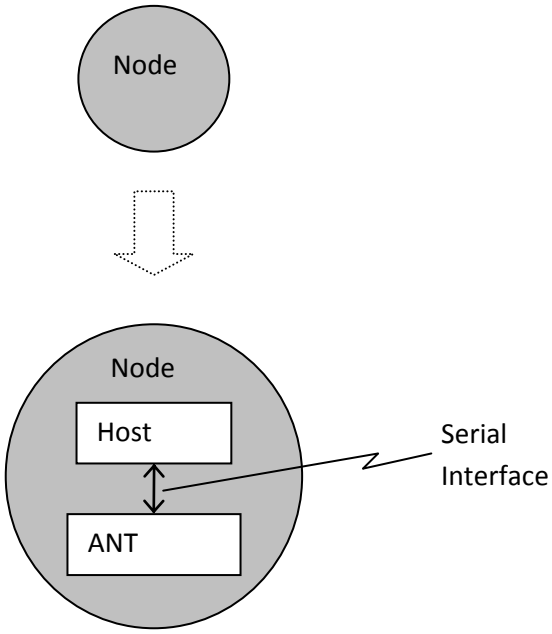


Figure 4-1. Contents of an ANT node

5 ANT Channels

In this section, further details are presented about the ANT protocol's most fundamental building block: the channel. As previously discussed, a channel must be established to connect two nodes together.

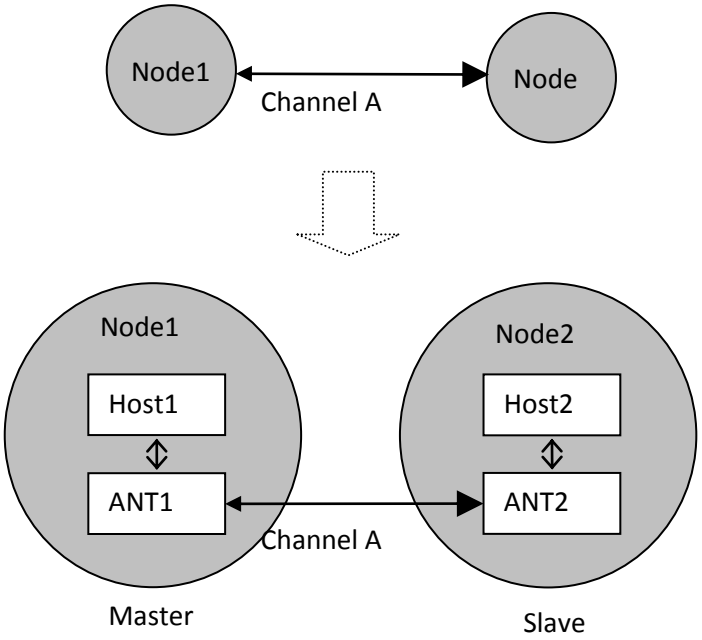


Figure 5-1. Channel communication between two ANT nodes

A channel consists of:

1. A master (e.g. Node1)
2. A slave (e.g. Node2)

5.1 Channel Communication

Communication between ANT nodes takes place in different ways depending on several factors including: what channel type is used; how that channel is configured; what data type is being sent; and which direction the data is sent in.

The majority of ANT implementations use synchronous, independent, bidirectional channels. When a master node opens a synchronous channel, the master node will first open a search window to check that its transmission is not likely to interfere with the transmission of another device, and will then transmit messages at the designated channel period (T_{ch}). In other words, once the channel is opened, a master device will always transmit a message on each channel timeslot as shown in Figure 5-2. When using bidirectional channels, the master device keeps its radio receiver on for a short time after it has transmitted each message. This allows the slave to optionally send data back to the master immediately after it has received a message.

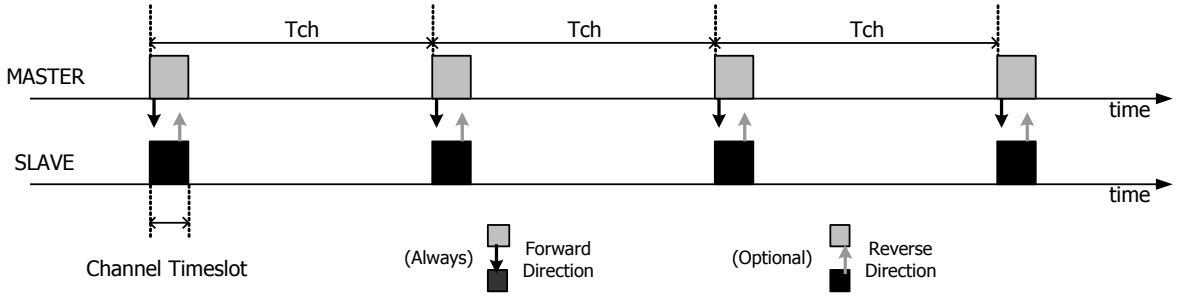


Figure 5-2. Channel communication showing forward and reverse directions. Not to scale.

The available channel types are listed and described in section 5.2.1. Each channel type must also be configured with the desired channel parameters (e.g. RF frequency, channel period and channel ID) and any additional features such as single channel encryption.

The ANT data types determine the way that the data will be sent between the two nodes of an ANT channel and are described in section 5.4. There are four data types: broadcast, acknowledged, burst and advanced burst message transfers. Each time the host application sends a data message to ANT for transmission, it specifies the data type along with the message data. Details on the host to ANT interface and messaging will be described in later sections.

Data messages are transferred between nodes in one of two directions:

1. Forward Direction (Master -> Slave)
2. Reverse Direction (Slave -> Master)

All data types can be transmitted in both directions, except across transmit/receive only channels.

5.2 Channel Configuration

In order for two ANT devices to communicate, they require a common channel configuration that includes information related to the operating parameters of a channel. The following information is required to define a channel configuration.

- Channel Type (section 5.2.1)
    - Optional Extended Assignment (section 5.2.1.4)
- RF Frequency (section 5.2.2)
- Channel ID (section 5.2.3)
    - Transmission Type (section 5.2.3.1)
    - Device Type (section 5.2.3.2)
    - Device Number (section (5.2.3.3)
- Channel Period (section 5.2.4)
- Network (section 5.2.5)

Although the configuration of a specific channel can remain constant throughout its connection, most parameters may be changed while the channel is open. Also, it should be noted that a master can maintain multiple channels that differ in terms of channel configuration parameters. Further information on which channel parameters must be set prior to opening a channel, may or may not be changed during an open channel, and resulting implications, can be found in section 5.3.

5.2.1 Channel Type

Channel type specifies the type of communication that will occur on the channel. It is an 8-bit field with certain acceptable values in the range of 0 to 255. The channel type must be specified prior to opening and establishing a channel. Some common channel types are given in Table 5-1.

Table 5-1. ANT channel types

Value	Description
0x00	Bidirectional Slave Channel
0x10	Bidirectional Master Channel
0x20	Shared Bidirectional Slave Channel
0x30	Shared Bidirectional Master Channel
0x40	Slave Receive Only Channel (diagnostic)
0x50	Master Transmit Only Channel (legacy)

5.2.1.1 Bidirectional Channel

For a bidirectional channel type, data can flow in both the forward and reverse directions. The primary direction data flow is determined by the node specified. For example, if a node establishes a bidirectional slave channel type, it will primarily receive but can still transmit in the reverse direction. Similarly, the master node will primarily transmit data in the forward direction but can also receive in the reverse direction. Please refer to section 5.1 for more information on the concept of forward and reverse data flow.

5.2.1.2 Shared Bidirectional Channel

Shared channels expand on the basic bidirectional channel types. Shared channels can be used where a single ANT node must receive, and possibly process, data from many nodes. In this scenario, multiple nodes will share a single independent channel to communicate with the central node. An example of a shared channel network is provided in Figure 5-5. Refer to sections 5.5 and 5.6 for more information regarding independent and shared channels respectively.

5.2.1.3 Transmit/Receive Only Channel

Transmit/receive only channels can only send data in the forward direction. In other words, the master cannot receive data from any slave. Similarly, the slave can only receive data, the slave cannot send data. As such, this channel type can only use the broadcast data type (described in section 5.3) and should not be used if the application requires any form of confirmation or acknowledgement of the successful receipt of data. The transmit/receive only channel type exists for legacy support and is not recommended for general use as it also disables the ANT channel management mechanisms. Receive only channels are recommended for diagnostic applications using continuous scan mode.

5.2.1.4 Channel Extended Assignment

The optional extended assignment byte allows various ANT features to be enabled. Currently, these features are frequency agility, background scanning, fast channel initiation, and asynchronous transmission. The extended assignment byte is not available on all ANT devices; please refer to datasheets for more details.

5.2.1.4.1 Frequency Agility

Similar to frequency hopping schemes, ANT frequency agility allows a channel to change its operating frequency to improve coexistence with other wireless devices such as Wi-Fi. However, unlike frequency hopping, this functionality will monitor the channel's performance and only change operating frequencies when significant degradation is observed. Both the master and the slave must be configured with frequency agility enabled, and have the same three operating frequencies set.

For more information refer to the "ANT Frequency Agility" application note. This application note also explains how to implement frequency agility at the application level for those ANT devices that do not have frequency agility as a built in feature.

5.2.1.4.2 Background Scanning

A receive channel configured for background scanning performs a continuous search operation. As with standard ANT search, it can be performed in either high or low priority modes. Only one channel configured for background scanning should be open at a time on a single device. For more information refer to the "ANT Channel Search and Background Scanning" application note.

If other channels are open, it is recommended that the background scanning search timeouts are configured for low priority search mode. This will ensure that the background scanning search mechanism does not interfere with any other channels operating on the device.

Background scanning can also be used in conjunction with proximity search. See section 6.2 for more details.

5.2.1.4.3 Fast Channel Initiation

Opening a channel that is configured for fast channel initiation will start a synchronous channel as soon as possible and will skip the search window check that is otherwise performed prior to starting a synchronous channel. This allows for channels to reduce the latency between the open command and transmission. This function is useful in scenarios where the device opens a channel for brief periods of time and requires a low latency at start up e.g. control applications.

Configuring a channel for fast channel initiation is achieved in a similar way to configuring other channel features. The fast channel initiation attribute is assigned using bit 4 of the extended assignment field of the assign channel message.

Data sent from this channel should not block currently active synchronous channels. If another active channel is currently transmitting a message (broadcast, acknowledge or burst), then the fast channel will attempt to send data as soon as the other channel has released the radio. This will result in some start up latency.

Fast channel initiation is not recommended for use in group environments. This is because skipping the search window check, increases the risk that the new channel will transmit at the same time as an existing device and cause interference between the two devices.

5.2.1.4.4 Asynchronous Transmission

The purpose of asynchronous transmission is to transmit data over-the-air only when required by the application. This is in contrast to normal master devices that transmit data at regular channel periods. This type of channel is useful for remote control type of applications where it is beneficial to send data immediately after a user generated event, such as a button push. It is recommended that a channel configured for asynchronous transmission should be paired with a channel opened in continuous scanning mode on the receiving end (section 5.7). As such asynchronous transmission is appropriate for use cases where the receiving end has system power available to support the power requirement for continuous scanning. As asynchronous transmissions only occur when there is new data to send, transmitting devices can sleep for extended periods of time, drastically reducing power consumption. This channel configuration is **not** appropriate for applications that require streaming data or low power on both sides of the link.

To configure a channel for asynchronous transmission, the asynchronous attribute is assigned using bit 5 of the extended assignment field of the assign channel message. In contrast to other channel configurations, a channel using asynchronous transmission does not need to be opened and no channel period needs to be set. Once the channel is assigned any data messages pushed to the channel will be sent over the air as soon as a free transmission interval is available using the assigned RF frequency and network key. All three data types (broadcast, acknowledged and burst) are supported.

Asynchronous data will not block currently active synchronous channels unless the transmission on the asynchronous channel is a burst. This may result in latency sending data out after a data message is received. Sending data to a channel set up for asynchronous transmission should not disrupt a burst already in progress on another channel.

5.2.2 RF Frequency

ANT technology supports the use of any of the available 125 unique RF operating frequencies. When assigning frequencies, it is important to check for compliance with international standard frequencies. A channel will operate on a single frequency throughout its existence, unless manually changed by the controlling application. The channel frequency must be known and adhered to by both master and slave prior to the establishment of a channel. After the channel has been established, the RF frequency can be changed "on the fly" (i.e. while the channel is open); however, the new frequency must be set at both the master and the slave nodes. Note that this can result in the slave node returning to search mode until it finds, and synchronizes with, the master.

The RF frequency is an 8-bit field with acceptable values ranging from 0 to 124. This value represents the offset in 1MHz increments from 2400MHz, with the maximum frequency being 2524MHz. Equation 5-1 can be used to determine the value for the RF frequency field.

$$RF\_Frequency\_val = \frac{Desired\_RF\_Frequency(MHz) - 2400MHz}{1MHz}$$

Equation 5-1. Channel RF frequency

For example, if a network operating frequency of 2450MHz was desired, the RF frequency field will be set as 50.

The default RF frequency field value is 66 and represents the network operating frequency of 2466MHz.

It is important to note that it is not necessary to use different RF frequencies to support multiple coexisting channels. The TDMA nature of the ANT system means that a large number of channels can coexist on a single common RF frequency. It is the product developer's responsibility to ensure that RF frequencies used will comply with the regulations of all regions of the world in which this equipment is to be used.

5.2.3 Channel ID

The most basic descriptor of a channel, and one that is crucial in device pairing, is the channel ID. In order to establish an ANT channel, the host must specify its channel ID (if master), or the channel ID it wishes to search for (if slave). It's a 4-byte value that contains 3 fields – Transmission Type, Device Type (including pairing bit) and Device Number. For a private or a public network, these three fields can be user defined. Typically, the device type is a number that represents the class (or type) of the master device. The device number is a unique number representing a specific master device. The transmission type is a number that represents the different transmission characteristics of a device, which can be determined by manufacturer or pre-defined in an ANT+ (or any) managed network.

Only devices with matching channel IDs can communicate with each other. The channel ID represents the device type/number and transmission type of the master device and must be specified on the master device. On a slave device, these fields are set to determine which master device to communicate with. They can be set to match a specific master, or any/all of these fields can be set to zero, representing a wildcard value, such that the slave will find the first master matching other channel parameters (network key, frequency).

The channel ID may be changed at any time. However caution must be taken to ensure that wildcards are not used when changing a channel ID while a channel is open.

The three types are described in more detail in the following sections.

5.2.3.1 Transmission Type

The transmission type is an 8-bit field used to define certain transmission characteristics of a device. Specifically, the two least significant bits of the transmission type are used to indicate the presence, and size, of a shared address field at the beginning of the data payload, and the third least significant bit (LSB) is used to indicate the presence of a Global Data Identification Byte (such as ANT+ page numbers). For details on shared channels refer to section 8.2

The most significant nibble may optionally be used to extend the device number from 16 bits to 20 bits. In this case, the transmission type most significant nibble becomes the most significant nibble of the 20 bit device number.

This parameter must be specified on a master device; however, it can be set to zero (wildcard) on a slave device.

On private and public networks, the transmission type can be defined as desired; however the lower 2 bits still represent the shared address. On ANT+ managed networks (i.e. the ANT+ and ANT-FS networks) the following definitions apply:

Table 5-2. Transmission Type Bit Field

Bits	Description
0-1	00:Reserved 01: Independent Channel 10: Shared Channel using 1 byte address (if supported) 11: Shared Channel using 2 byte address
2	Optional for non-ANT+ managed networks: 0: Global data pages not used 1: Global data pages used (e.g. ANT+ Common Data pages)
3	Undefined – set to zero.
4-7	Optional extension of the device number (MSN)

5.2.3.2 Device Type (including Pairing Bit)

The device type is an 8-bit field used to denote the type (or class) of each participating network device. This field is used to differentiate between multiple nodes of network devices such that participants are aware of the various classes of connected nodes and can decode the received data accordingly. For example, one device type value could be assigned to heart rate monitors, which will be different to the value assigned to bike speed sensors, and their respective data payloads will be interpreted accordingly.

Please note that the most significant bit of the Device Type is a device pairing bit. Refer to section 6, and the "Device Pairing" application note for more information on device pairing.

This parameter must be specified on a master device; however, it can be set to zero (wildcard) on a slave device. For private networks, the device type can be defined as desired. Specific implementation-level information about channel ID usage is provided in the channel ID functional description in section 9.5.2.3.

5.2.3.3 Device Number

The device number is a 16-bit field that is meant to be unique for a given device type. Typically, this may be correlated to the serial number of the device, or it could be a random number generated by the device if the process of setting serial numbers for a particular product is unavailable. This parameter must be specified on a master device, i.e. it cannot be set to zero. In a slave device, this field may also be used as a wild card during device pairing as described in the previous section. The channel ID functional description is located in section 9.5.2.3. Note that if the serial numbers ending in 0x0000 or 0xFFFF should not be used, as these are reserved values.

5.2.4 Channel Period

The channel period represents the basic message rate of data packets sent by the master. By default, a broadcast data packet will be sent (master) and received (slave), on every timeslot at this rate. The channel message rate can range from 0.5Hz to above 200Hz, with the upper limit dependant on the specific implementation.

The channel period is a 16-bit field with its value determined by Equation 5-2.

$$Channel\_Period\_val = \frac{32768}{MessageRate(Hz)}$$

Equation 5-2. Channel period

For example, to have a message rate of 4Hz on a channel, the channel period value must be set to 32768 / 4 = 8192.

The default message rate is 4Hz, which is chosen to provide robust performance as described below. It is recommended that the message rate be left at the default to provide more readily discoverable networks with good power and latency characteristics.

The maximum message rate (or the minimum channel period) depends on the computational capacity of the system. High data rates in combination with multiple active channels will substantially limit the maximum message rate.

Bursting, which is described in the following section, can achieve a data rate of 20kbps. This is independent of the message rate. In other words, the message rate will affect the time between whole burst transfers, but does not affect the actual rate of bursting.

Proper assignment of channel period is critical and it is imperative to be mindful of the following issues:

- The message rate is directly proportional to the power consumption. Please see respective ANT product datasheet for details.
- A small channel period allows for higher data-transfer rates.
- A small channel period results in faster successful device-search operations.

5.2.5 Network

ANT supports the establishment of numerous unique public, managed and private networks. A particular network may specify a set of operating rules for all participating nodes. In order for two ANT devices to communicate, they must be members of the same network. This provides the ability to establish a network that can be publicly available, or purposely shared among multiple vendors with the goal of establishing an 'open' system of interoperable devices.

A managed network defines rules and specific behaviours governing its use. An example of a managed network is the ANT+ network. Those companies who have adopted the ANT+ promise of interoperability are members of the ANT+ Alliance, a special interest group which fosters optimized brand value and partnerships with other top tier products. The key advantage of this unique managed network is device specific interoperability which enables wireless communication with other ANT+ products. Target applications include any wireless sensor monitoring of sport, wellness or home health.

ANT+ has device profiles that specify data formats, channel parameters and the network key. Examples of ANT+ Device Profiles include:

- Heart rate monitor
- Speed and distance monitors
- Bike speed and cadence sensors
- Bike power sensor
- Weight scale (for example, tracking BMI and percent body fat)

- Fitness equipment data sensors
- Temperature sensor

Conversely, a private network could be defined to ensure network privacy and restrict access to intended participating devices only. Channels can be independently assigned to different networks so that it is possible for a single ANT device to be a member of multiple networks.

The ANT Network has two components which are described below.

5.2.5.1 Network Number

A network number is an 8-bit field that identifies the available networks on an ANT device, with acceptable values ranging from 0 to the maximum number defined by the ANT implementation. The host can obtain this maximum number by querying the ANT system using the appropriate request message (refer to section 9 for more details). The default network number is 0. Network number 0 is assigned to the "Public Network" by default. For AP1 devices, the remaining network numbers are left un-initialized; however, for non-AP1 devices all network numbers typically default to the public network key.

The network number will be assigned a network key using the Set Network Key (0x46) message (refer to section 9.5.2.7), and any individual channel assigned to a network number will use the associated 8-byte network key. Multiple channels can be assigned to the same network number, so a network key can be used in multiple channels without having to enter the key multiple times.

5.2.5.2 Network Key

The network key is an 8-byte number that uniquely identifies a network and can provide a measure of security and access control. The network key is configurable by the host application and a particular network number will have a corresponding network key. Only channels with identical valid network keys may communicate with each other. Also, only valid network keys will be accepted by ANT. If a Set Network Key (0x46) command is sent with an invalid key, the network key will not be changed; it will retain the value it held prior to the command.

The network number and the network key together provide the ability to deploy a network with varied levels of access control and security options. By default, ANT firmware assigns the network number 0 with the default public network key. This network is open to all participating devices and has no set rules governing its use.

For more information on established public/managed networks or initiating your own network, please contact Dynastream at www.thisisant.com.

5.2.6 Example Channel Configuration

An example channel configuration for a simple application is given in Table 5-3.

Table 5-3. Example channel configuration

Parameter	Value	Description
Network Number	0	Default Public Network
RF Frequency	66	Default Frequency 2466MHz
Device Number	1	Sample Serial Number
Transmission Type	1	Transmission Type (no shared address)
Device Type	1	Sample Device Type
Channel Type	0x10	Bidirectional Transmit Channel
Channel Period	16384	2Hz Message Rate
Data Type	0x4E	Broadcast

Note the network number is set to '0'; this is the default network number for the public network key.

5.3 Establishing a channel

The prerequisite for establishing a channel is that the master and slave must have common knowledge of the channel configuration as outlined in section 5.2. Figure 5-3 illustrates the process required to properly establish communication between two ANT nodes. Certain channel parameters (within solid lines) have no default value and must be set by the application, while other parameters (within dashed lines) do have defaults and only require setting if a different value is desired.

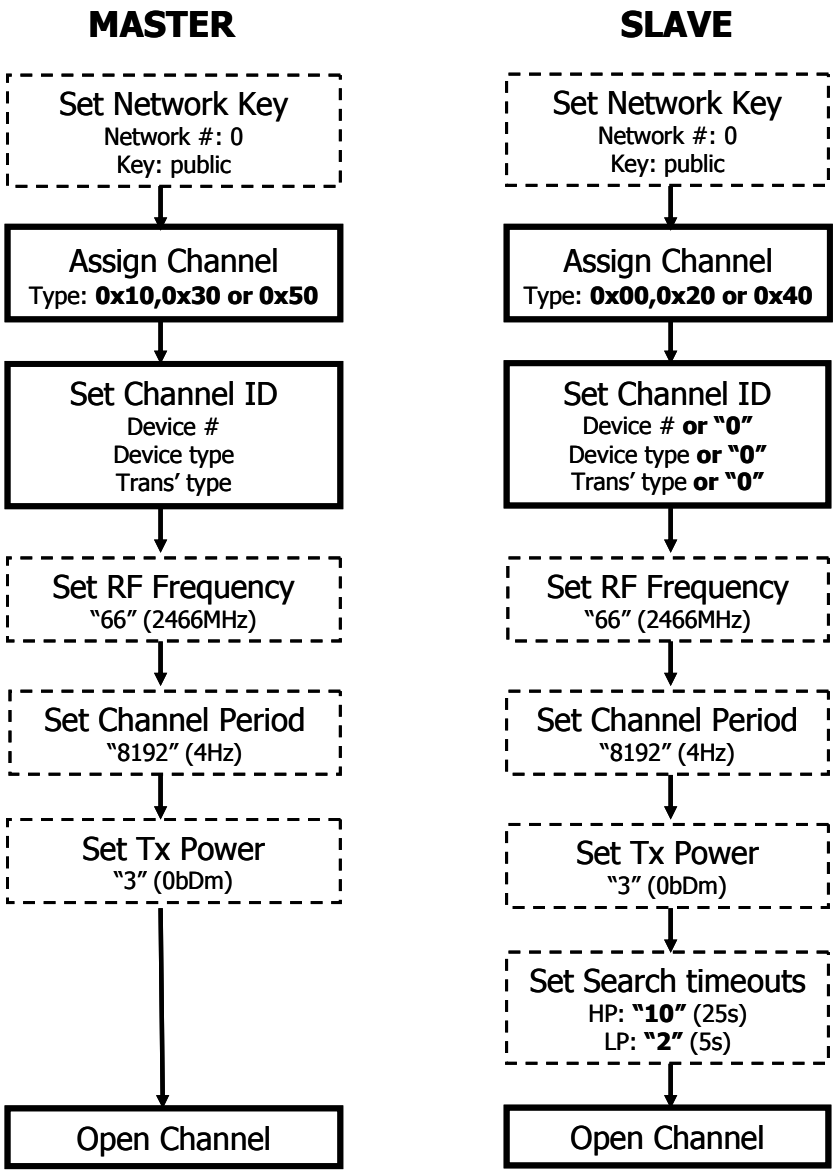


Figure 5-3. Process to establish a channel between master and slave nodes.

The default network configuration is the public network key, assigned to network number 0. If a private or managed network is desired, the appropriate network key must be assigned to a network number, which can then be assigned to a channel. Refer to section 9.5.2.7 Set Network Key (0x46) for details.

After (optionally) setting the network key, the channel type must be assigned to the channel you wish to open. For example, the master node will need to be assigned one of the transmit channel types, and the slave node assigned a corresponding receive channel type. **Once the assign channel command has been sent, most other channel**

parameters will return to their default values (RF Frequency, Channel Period, Set Tx Power, Set Search Timeouts). Refer to section 9.5.2.2 Assign Channel (0x42).

Next, the Channel ID must also be set. The device number/type and transmission type must be specified on the master node. The slave can set all, some or none of these fields to match those of the master depending on the application. Any field that does not match that of the master should be set to a wildcard value of zero. Refer to section 9.5.2.3 Set Channel ID (0x51).

If desired, other channel parameters such as RF frequency (section 9.5.2.6), Channel period (section 9.5.2.4), the yet to be discussed Tx power (section 9.5.2.8), and search timeouts (sections 9.5.2.5 & 9.5.2.14) can also be set, but are not required.

The final step is to open the channel (section 9.5.4.2). Once opened, the master establishes the channel by transmitting 8-byte data packets in the designated timeslot at the established message rate. The master ANT channel will be maintained indefinitely at this message rate. The channel master's host controller will optionally provide new data to the ANT engine for continuing transmissions.

The slave on the other hand, will immediately start searching for a master that matches the channel ID criteria. Once the master has been located, and a connection established, the slave receives data indefinitely at the given message rate. If no master is found within the given timeout periods, then the slave channel will close. As the master never searches, no timeout values need to be set. The master will transmit until the channel is specifically closed by the application.

5.4 ANT Data Types

There are four data types supported by ANT: broadcast, acknowledged, burst and advanced burst data. Each data type is sent in 8 byte packets over the RF channel. The data type is not a channel configuration parameter and a bi-directional ANT channel is not restricted to a single data type. In other words, any of the four data types can be sent in either the forward or reverse direction, at the channel's designated timeslot, at the discretion of the host. The only restriction is for unidirectional channels, which can only send broadcast data in the forward direction.

5.4.1 Broadcast Data

Broadcast data is the most basic data type and is the system default. Broadcast data is sent from the channel master to the slave on every channel timeslot. Broadcast data is only sent from the slave to the master in the reverse direction if expressly requested by the slave's Host MCU (by default, no data is sent without a request).

A master device is always transmitting in the forward direction, at every timeslot. As stated earlier, the broadcast data type is the system default. If no new data has been provided by the host, the previous message packet, whether it was sent broadcast or otherwise, will be re-transmitted as a broadcast message. Messages in the reverse direction, on the other hand, are not required on each channel period. As such, broadcast messages are only sent in the reverse direction once.

Broadcast data is never acknowledged, so the originating node will not be aware of any lost data packets. In the case of a one-way transmission link (i.e. transmit-only master communicating to a slave), broadcast data is the only available data type due to the inability of the master to receive an acknowledgement.

Broadcast data consumes the least amount of RF bandwidth and system power. It is the preferred choice of communication where occasional data loss is tolerated (although it should be noted that any data loss will be very limited in most non-hostile RF environments). An example system where occasional data loss is not critical is that of a temperature logging system, where changes in temperature are relatively slow compared to the communications message rate.

5.4.2 Acknowledged Data

At any time during an established bidirectional connection, in either the forward or reverse direction, a device can choose to send an acknowledged data packet at the next timeslot. The node that receives the acknowledged data packet will respond with an acknowledgment message back to the originating device. The host controller of the originating device will be notified of that packet's success or failure, therefore knowing that the packet transmitted successfully. There is no automatic re-transmission of unacknowledged data packets.

The master host application may send every data packet as acknowledged data, or may mix broadcast and acknowledged data as appropriate to the particular application. To decide which is more appropriate, the following should be taken into consideration:

- Acknowledged data packets use more RF bandwidth and consume more power, which should be taken into consideration when designing power-sensitive applications.
- Acknowledged data is ideally suited for the transmission of control data, ensuring that both nodes are aware of each other's state.

For a master device, if the data type isn't specified as acknowledged or, if an acknowledged message was sent and no new data provided before the next transmit time slot; the message is sent as Broadcast data type on the next channel time slot.

5.4.3 Burst Data

Burst data transmission provides a mechanism for large amounts of data to be sent between devices. Burst transfers consist of a rapid series of continuous acknowledged data messages. The rate at which packets are burst across the channel is independent of, and significantly faster than, the channel period; resulting in a maximum 20kbps data throughput. It should be noted that this also means the burst packets are synchronized relative to each other, instead of to the regular channel period.

Similar to acknowledged messages, the originating host's MCU will be informed of the burst transfer's success or failure. However, the success/failure notification is for the entire burst transfer rather than for each packet and, unlike acknowledged messages, any lost data packets in the transfer will be automatically retried. Should any packet fail to transmit successfully after five retries, ANT will abort the burst transfer and notify the host MCU with a failure message.

When a single packet burst is sent, it behaves identically to an acknowledged message, and there are no retries associated with a single packet burst.

There is no limit on the duration of a burst transaction. However, burst transactions take precedence over all other open channels on both participating nodes. If there are other channels in the system, care should be taken to service them with reasonable frequency. Although the ANT protocol is robust and can handle outages caused by burst transfers or other external interference, excessive channel starvation may lead to loss of synchronization or data. An example of this is:

During a prolonged burst, as the packets are synchronized off each other, clock errors may cause the regular channel periods to drift, potentially losing synchronization. Once the burst completes, the channels are no longer synchronized and the slave drops into search.

Another extreme example of this would be if the master node of one channel was servicing a prolonged burst on another channel; if the burst duration was too long, the slave node of the former channel could lose synchronization, drop back into a search and timeout (closing the channel).

Bursting can create interference for other devices that are operating at the same RF frequency.

For more information on burst transfers please refer to the "Burst Transfers" application note.

5.4.4 Advanced Burst Data

Some ANT devices also support advanced burst transfer (as listed in section 9.4), which increases the maximum data throughput to 60kbps. Advanced burst effectively increases data throughput on burst transfers of 24 bytes or greater. For smaller bursts it is recommended to use standard burst data messages. Advanced burst transfer is described in section 9.5.2.25.

5.4.5 Summary of Data Types

Further details and sequence diagrams related to all data types are provided in section9.5.5. The host application software on both the master and slave sides should be implemented to expect common data types (i.e. broadcast vs. acknowledged vs. burst) to be utilized as appropriate for a particular application. The specific format of the contents of the data payload must be previously established by both host controllers such that data can be properly decoded and interpreted.

Table 5-4. ANT data types

Data Type	Channel Direction	Description
Broadcast	Forward	Default Data Type. Broadcast messages sent every timeslot (unless otherwise requested) and will be retransmitted if ANT has not received any new data from the master's host MCU
	Reverse	Broadcast messages optionally sent each channel timeslot. Only sent if specifically requested by the slave's host MCU. The message will be sent immediately after a message is received from the master i.e. it cannot be sent if no message is received. Sent only once, there is no retransmission
Acknowledged	Forward	If requested, sent on the next channel timeslot If the data type isn't specified as Acknowledged or if no new data is provided before the next transmit time slot, the message is resent as Broadcast data type on the next channel time slot
	Reverse	Acknowledged data types only sent when specifically requested by the slave's host MCU. The message will be sent immediately after a message is received from the master i.e. it cannot be sent if no message is received. Sent only once, there is no retransmission
Burst	Forward	A burst transfer will commence at start of the next timeslot. Bursts packets synchronize off each other. The last packet of the burst will be retransmitted on the next channel period if ANT has not received any new data from the master's host MCU
	Reverse	Burst data types only sent when specifically requested by the slave's host MCU. The message will be sent immediately after a message is received from the master i.e. it cannot be sent if no message is received. Not re-transmitted
Advanced Burst	Forward	An advanced burst transfer will commence at start of the next timeslot. Bursts packets synchronize off each other. The first packet of the burst will be retransmitted on the next channel period if ANT has not received any new data from the master's host MCU
	Reverse	Burst data types only sent when specifically requested by the slave's host MCU. The message will be sent immediately after a message is received from the master i.e. it cannot be sent if no message is received. Not re-transmitted

All data types can also be 'extended' such that the receiving node's ANT will pass additional information, along with the data, to the host. For more information see section 7.1.1.

5.5 Independent Channels

An independent channel has only one master and one slave. It is possible for the master or slave to be a master or slave to another, or a number of other, nodes. However, from the point of view of an independent channel, there is only one of each. For example, consider the four-node network in Figure 3-2. Each channel has only one master and one slave.

A broadcast network, shown in Figure 3-1, is also formed using independent channels even though the data from one master is received by many slaves. Such a network has a unique master who doesn't purposely initiate communication with multiple slaves on the same channel. Note that the data in a broadcast network is predominantly sent in the forward direction. This reduces the chance of multiple slaves simultaneously sending data to a single master. This is different from a shared channel, which has a single master and multiple slaves; however, there is an addressing scheme that allows for data flow in both directions (refer to section 5.6).

Although independent channels offer simplicity in implementation, a node can support a limited number of simultaneous independent channels within the confines of the system's computational ability. For example, the nRF24AP1 supports 4 independent channels.

For an implementation example using independent channels, refer to section 8.1.

5.5.1 ANT Single Channel Encryption

Single Channel Encryption can be enabled on one independent channel on supported devices; it cannot be applied to shared channels. Encrypted channels help to both enable and simplify use cases which require secure over the air communication e.g. medical devices, text communication, etc. Some ANT devices include single channel encryption as a feature that may be assigned to a channel. In these devices, encryption is handled at the protocol level, reducing the burden on the application layer.

Any number of ANT slaves may pair to an encrypted master channel. Once paired, devices that support ANT single channel encryption can negotiate with the master. Successful negotiation will enable the slave to decrypt the messages it receives from the master, and any future messages that the slave sends to the master will be sent as encrypted messages. Any listening device that does not support encryption, or that fails negotiation, will not be able to decrypt the data.

Further information about how to use ANT single channel encryption is available in section 9.5.2.30.

5.6 Shared Channels

Shared channels can be used where a single ANT node must receive, and possibly process, data from many nodes. In this scenario, multiple nodes will share a single independent channel to communicate with the central node. An example of a shared channel network is provided in Figure 5-5.

Shared channels are made possible by the use of a one- or two-byte Shared Channel Address field and a specific value for the Channel Type; both controlled by the host application. As will be detailed in a later section, ANT has an 8 byte data payload. The Shared Channel Address field replaces the first one or two bytes of the data payload as shown in Figure 5-4.

Independent Channel Data Payload

Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7

Shared Channel 7 Byte Data Payload

Shared Address	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6

Shared Channel 6 Byte Data Payload

Shared Address LSB	Shared Address MSB	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5

Figure 5-4. Independent and 1 or 2-byte shared channel data payloads.

If a channel is defined as shared, the host application provides ANT with the shared address and data. For example, with 2-byte addressing, more than 65k slave devices can share a single ANT channel, while a 1 byte shared address allows for 255 slave devices.

In a shared channel, the node that is intended to communicate with many other nodes must initiate the channel as the master. All other nodes that access this shared channel must be configured as slaves. All nodes, both master and slaves, must be configured as shared channels, have matching channel IDs (wildcards can be set on slaves when opening a channel, but will match upon a successful search), RF frequencies and channel periods. The master's host application must be aware of each slave node's address, and similarly, each slave's host application must also know its own shared address.

The master controls the communication by transmitting data at the channel message period. The master's host application will provide the data payload, including the shared address field as shown in Figure 5-4. All slaves on the channel will synchronize off this transmitted message; **however, ANT will only release the data to the slave's host if the shared address field matches the shared address for that node or if the shared address holds a value '0'**. The master can send data to all slaves at the same time using the Shared Channel Address of 0. A slave will respond in the reverse direction only if its Shared Channel Address matches the one it receives from the master. An example 2 byte shared address shared channel is shown in Figure 5-5, with master node M, and four slave nodes addressed 1:4.

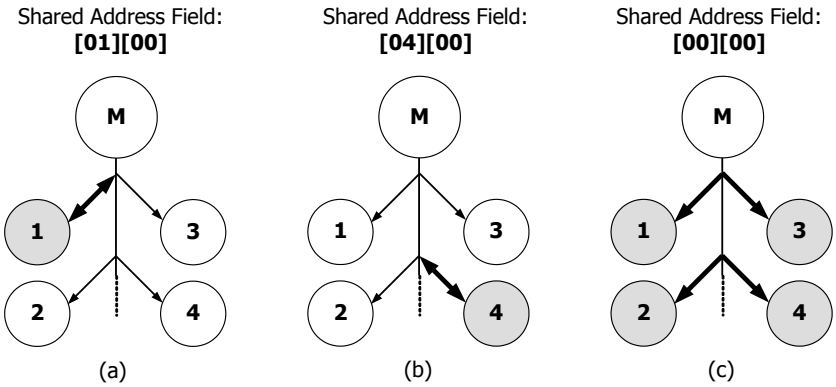


Figure 5-5. Example Shared Channel.

Grey nodes in Figure 5-5 indicate the node's host received data from ANT. The arrows indicate the direction of data flow.

Figure 5-5 a: The master's (M) host provided [01][00] in the shared address field (LSB MSB). ANT will transmit the data with this shared address on the next channel period. All slave nodes receive and use this message to maintain synchronization, but **only slave node 1**'s host will actually receive the data. The ANT protocol will prevent the data from progressing to an incorrectly addressed node's host. Slave node 1 has the option of sending data back to the master (i.e. in the reverse direction) at this time. No other slave node can transmit data to the master.

Figure 5-5 b: The master's host provided [04][00] shared address field. Similarly, the data is transmitted on the next channel period; all slaves use this transmission for synchronization; only node 4's host receives the data and has the option of transmitting in the reverse direction.

Figure 5-5 c: Master's host provides [00][00] in the shared address field. This indicates a broadcast to ALL nodes. As such, each slave host receives the data. There is no reverse direction when broadcasting to all slaves, therefore no slaves can transmit.

The shared channel concept is extensible to acknowledged data and burst data transactions. In burst data transactions, only the 1st data packet requires the Shared Channel Address in the data payload, the remaining data packets may contain only the application data.

Please refer to section 8.2 for a sample network implementation and to see the sequence of commands required to create a shared channel.

The shared channel functionality can also be extended for 'ad hoc' joining/leaving of channel by implementing an auto shared channel. For more information see application note "Auto-Shared Channel".

5.7 Continuous Scanning Mode

Continuous scanning mode is another method that can be used when a single ANT node must receive, and possibly process, data from multiple nodes. Rather than a single master controlling multiple slaves (as for shared channels) a node in continuous scanning mode receives full-time, allowing it to receive from multiple transmitting masters at any time. Similar to a shared channel, all devices operate on the same RF frequency.

The ANT radio on the central node is always occupied with the continuous scanning mode; hence, no other channels can be open on that node. Also, as the RF is continually active, this node draws significant power (approximately 3x the amount of a typical transmit channel) and therefore continuous scanning mode should not be used for devices that have tight power constraints.

Each of the transmit nodes should have unique device numbers, such that its channel ID is also unique. With a unique channel ID, the central node can correctly attribute each received message to its corresponding master device.

The receiving node is configured as a bidirectional receive channel that is opened with the Open Rx Scan Mode (0x5B) command (refer to section 9.5.4.5). As the node is receiving full time, the channel period does not need to be set. Although the central node is receiving full time, it can transmit messages back to the master nodes. For this to happen, a master must first transmit to the receiving node, which can then optionally send data back to that specific master in the reverse direction.

A receive only channel type can be used in conjunction with the continuous scanning mode for diagnostic applications.

See the "Continuous Scanning Mode" application note for more details on implementing the continuous scanning mode.

In comparison to using a node in continuous scanning mode, shared channels have the advantage of maintaining low power at all nodes. However there is some latency due to the synchronous nature of the shared channel, and the time involved to service each individual node. As the central node in continuous scanning mode is always receiving, there is very little latency and, should the central device have sufficient power capabilities, this mode is advantageous when intermittent, asynchronous, or instantaneous transmissions are desired. Alternatively a channel can be configured for background scanning, which operates as described in section 5.2.1.4.2

Please note, not all modules can support continuous scanning mode; refer to section 9.4 for their respective capabilities.

6 Device Pairing

The act of pairing two devices (master with slave) involves establishing a relationship between two nodes that wish to communicate with one another. This relationship can be permanent, semi-permanent or transitory.

A pairing operation consists of a slave device acquiring the unique channel ID of the master device. If permanent pairing is desired, the slave node should store the master's ID in permanent or non-volatile memory. This ID will then be used to open a channel with this ID in all subsequent communication sessions. In semi-permanent relationship, the pairing lasts as long as the channel is maintained. Once it times out, the pairing is lost. In transitory, the pairing is temporary – for as long as is needed to get some data.

Please note that if a master uses only broadcast messaging, or if it uses the shared channel feature, multiple slaves may pair and communicate with the same master.

As previously mentioned, when the master device's channel is opened, it will start broadcasting messages. Its unique channel ID is broadcast with every message. When a slave device's channel is opened, it will immediately start searching for a master that matches the channel ID provided by the slave host MCU. In the case where a slave does not have knowledge of a specific master's channel ID, a pairing mechanism is available. The slave can search for a master using a wild card ID (value '0') in any, or all, of the channel ID fields. The slave will then search according to the criteria that it does know. For example, the slave may know what device type it wishes to connect to, but not the actual device number or transmission type. The slave's host application would then set the channel ID with the known device type, and place a wildcard (i.e. 0) in the remaining fields. On opening the channel, the slave would then search for any masters of that specific device type, and of any device type or transmission type; upon a successful search result, the specific ID of the master can be stored and used in the same manner as previously described for all future communications.

The pairing bit, which is the most significant bit (MSB) of the device type field, is an advanced pairing feature. On the slave side, the pairing bit is only checked by ANT if at least one of the fields of the Channel ID is a wild card. On the master side, the pairing bit must be set to indicate it is available for pairing.

Note that the pairing bit does not have to be set for pairing to occur; however, the status of the pairing bit must match for pairing to occur. This feature allows for more control. For example, a slave may have a fully wild-carded channel ID and the pairing bit not set. This would result in the slave searching for any broadcasting master. Alternatively, if the slave were to have the pairing bit set with a fully wild-carded channel ID then it would search only for a master that also had its pairing bit set. This is a somewhat simple example but illustrates how pairing can be aided via the pairing bit.

For more information see the "Device Pairing" application note and the examples that follow.

6.1 Pairing Example

An example pairing operation on a network of three remote temperature sensors (masters) and one base unit (slave) is shown below.

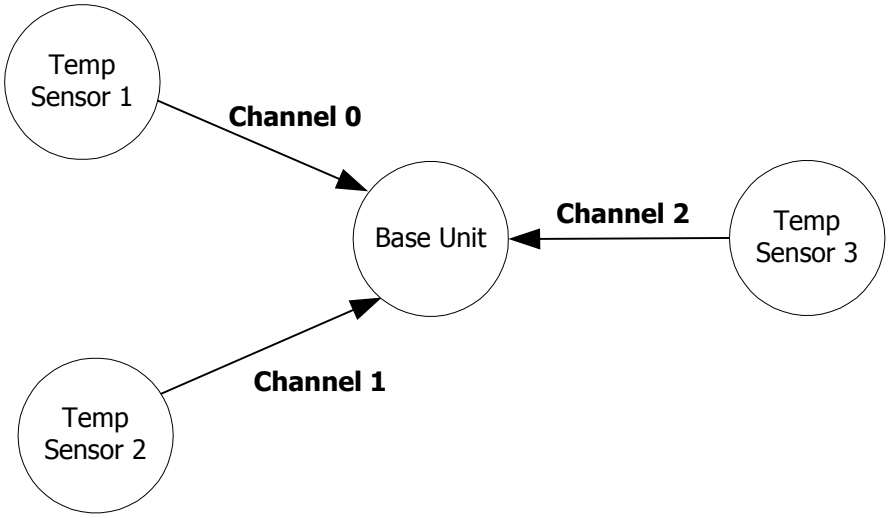


Figure 6-1. Example ANT network for use in device pairing

The base unit wishes to establish a permanent relationship with all temperature sensors. To initiate the pairing operation, each temperature sensor should be placed into a pairing mode. From a user perspective, it is left to the application to define the method of entry into pairing mode. For example this could be done upon initial insertion of a battery, or by means of a button push by the user, etc. As far as the ANT serial message interface is concerned, the host controller invokes a pairing mode by sending the following messages to the ANT engine (See section 9.3 for details):

1. Configure Channel
2. Set Channel ID (discoverable – i.e. device type=temperature sensor with pairing bit set)
3. Open TX Channel
4. Begin transmitting data on channel timeslot

At this time, the base unit (slave) must be prepared to search for the ID of the appropriate device type (temperature sensor). It performs the following:

1. Configure Channel
2. Set Channel ID (Transmission Type = Specific or Wild card, Device Type = Temperature sensor with Pairing Bit Set, Device Number = Wild Card)
3. Open RX Channel
4. Begin searching

The base unit finds a temperature sensor device type with pairing bit set. The channel is established, the slave ANT engine will pass the specific channel ID for that device to the host controller, which will store the ID for future channel establishment. This procedure is repeated for all three temperature sensors.

Each temperature sensor can choose to disable its discoverability after a time-out period (or after connection acknowledgement from the base unit if bidirectional transmission is supported) in order to be 'invisible' to future discovery by other slave devices.

This pairing process is required only once for the lifetime of an ANT system if a permanent relationship between two specific devices is desired. In such cases, device pairing may be performed during product manufacturing (factory environment) to remove burden from the customer.

6.2 Inclusion/Exclusion Lists

Another pairing feature available on some devices is the inclusion/exclusion list; refer to section 9.4 or request the device capabilities, section 9.5.7.4. For each available channel on a device, up to four channel IDs can be sent to the module and stored in that channel's inclusion/exclusion list.

When enabled and configured as an inclusion list, the channel IDs stored will be the only channel IDs accepted in a wild card search. This means that the slave will only connect to one of the specific master channel IDs listed (or in the case of a wild card, the first matching ID found). Similarly, if this feature is configured as an exclusion list, the slave will not acquire any master with a listed channel ID.

Note that any inclusion/exclusion lists should be cleared when the channel using the list is unassigned.

Not all ANT devices support inclusion or exclusion lists, and those that do may support the lists differently. In particular, some ANT devices require all fields to be defined with non-zero values (i.e. no wildcards), while other ANT devices do support wild card values (i.e. "0") in one or more of the channel ID fields.

Refer to sections 9.5.2.90, 9.5.2.11 and "Device Pairing" application note for more details.

6.3 White/Blacklists used with Single Channel Encryption

Whitelists and blacklists are very similar to inclusion and exclusion lists except:

- White/blacklists are used to exclusively allow or disallow encryption negotiation requests.
- White/blacklists are defined on the master node, whereas inclusion/exclusion lists are defined on the slave.
- Devices are screened based on their encryption ID rather than the channel ID

This feature is supported by all ANT devices that support single channel encryption. Refer to sections 5.5.1, 9.5.2.10, and 9.5.2.12 for further information.

Note that any white/blacklists should be cleared when the channel using the list is unassigned.

6.4 Proximity Search

Another feature to aid in device pairing is proximity search, which allows channels to be acquired according to the relative distance between two devices. In a standard ANT search, as described in the earlier section, the channel is opened and the slave device starts searching for a master with a matching channel ID. If any part of the channel ID is assigned a wildcard; then the slave could potentially match to one of a number of masters in range. For example, if a slave sets its device type to search for a specific kind of device (e.g. heart rate monitor), but placed a wildcard in all other fields of the channel ID, and there were four heart rate monitors in range (Figure 6-2 a, grey shading indicating slave's range), on opening its channel it could pair to any one of the four heart rate monitors depending on which transmitting master it found first.

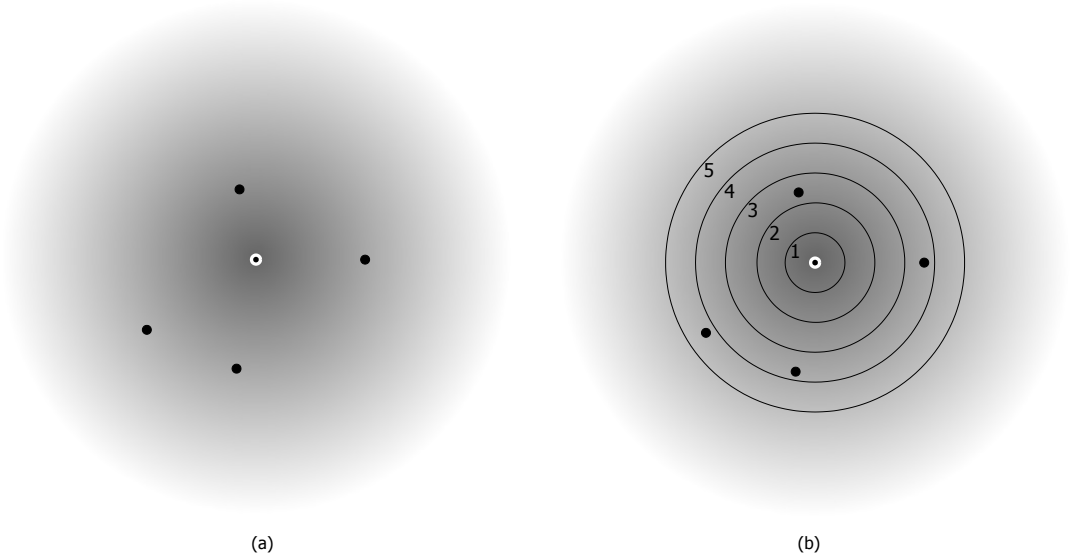


Figure 6-2. (a) Standard search (b) proximity search, showing bins 1-5 (of maximum 10).

Proximity search designates 'bins' of proximity ranging from 1 (closest) to 10 (furthest) as illustrated in Figure 6-2 b. The bins do not correlate to specific distances as this is very design-dependent (antenna design/orientation, etc.) and will need to be determined by the designer. Incremental distances are also design dependent.

The recommended use is to initially set the proximity search threshold value to bin 1 (Figure 6-3 a), as the smaller the search area, the better the results as far as limiting the possibility of finding the wrong device. Setting the threshold value too high could result in connection to one of multiple devices (Figure 6-3 b). Choosing an appropriate proximity threshold is critical in limiting the search accordingly and acquiring the desired device (Figure 6-3 c).

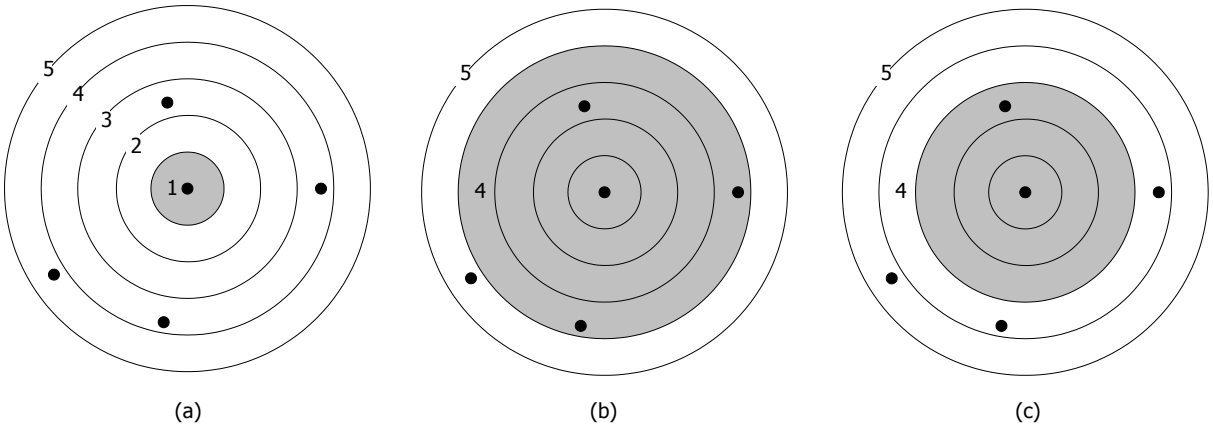


Figure 6-3. Varying proximity thresholds.

Proximity search can be used in conjunction with ANT searches and background scanning, but not with continuous scanning mode.

Proximity search is disabled by default. Once enabled, it is a one time requirement and the threshold value will be cleared upon a successful acquisition. If the search times out, or if using a background scanning channel, the threshold value is maintained

For more information please see the "Proximity Search" application note.

This feature is only available on certain ANT devices; please refer to section 9.4 or request the device capabilities (section 9.5.7.4).

7 ANT Interface

The host application and ANT typically communicate through a simple serial interface. The host can take the form of an embedded microcontroller or a PC, but the basic functionality remains unchanged. For more details, see the Interfacing with ANT General Purpose Chipsets and Modules document.

In the case of SoCs and mobile devices the interface to ANT is handled via libraries and the message framing described in section 7.1 below is not needed.

7.1 Message Structure

A typical serial message between the host and ANT engine has the following basic format.

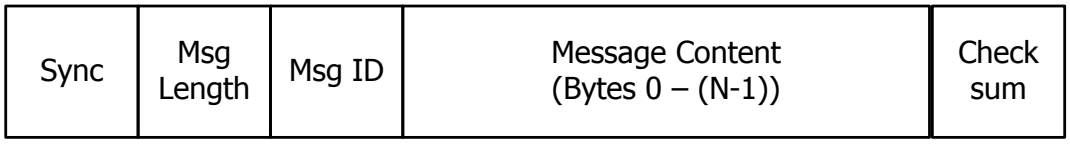


Figure 7-1. ANT serial message structure

As shown above, each message begins with a SYNC byte and ends with a CHECKSUM. The bytes are sent LSB first. Table 7-1 describes each component of the serial message shown above.

Table 7-1. ANT serial message components

Byte #	Name	Length	Description
0	SYNC	1 Byte	Fixed value of 10100100 or 10100101 (MSB:LSB) (Refer to the "Interfacing with ANT General Purpose Chipsets and Modules" document for details.)
1	MSG LENGTH	1 Byte	Number of data bytes in the message. (Refer to section 9.3)
2	MSG ID	1 Byte	Data Type Identifier 0: Invalid 1..255: Valid Data Type (See section 9 for details)
3..N+2	MESSAGE CONTENT	N Bytes	Content of the message as described in section 9.
N+3	CHECKSUM	1 Byte	XOR of all previous bytes including the SYNC byte

A complete summary of supported messages between a host and the ANT engine is presented in section 9. The table is valid for all types of ANT interface: Host MCU ⇔ ANT and Host PC Interface ⇔ ANT. Message formatting is first presented in summary form, which includes message length, ID and the message content of each respective message type. The message types are defined in section 9.1, and include Configuration, Notification, Control, Data, Channel Event/Response, Requested Response, Test Mode and legacy Extended Data messages.

Please note that the multi-byte fields have been implemented in little endian format. Using the example of a channel ID message, the least significant byte of 'Device Number' is assigned to byte 0, and the most significant byte to byte 1.

7.1.1 Extended Messages Format

Data type messages (detailed in section9.5.5) can be extended to allow ANT to pass additional information to the host, along with the received data message. There are two extended formats supported by ANT: flagged and legacy. Which format is used depends on the ANT device in use (refer to section 9.4). Later generation devices support the flagged extended messages format, AP1 does not support extended messages, and AT3 supports the legacy format as shown in Figure 7-2.

The extended data will be added to the data message as shown in Figure 7-2. Note the basic frame format of Sync, Message Length (ML), Message ID (ID) and checksum (CS) remains the same as described in Figure 7-1. However, instead of just the normal message content (channel number and 8-byte data payload), the host will now receive the message content followed by a flag byte (0x80) indicating the presence of extended data bytes. The message length value will be altered to account for

these additions. If extended messaging has been enabled, the message length and flag bytes must be checked to see if extended data bytes are present.

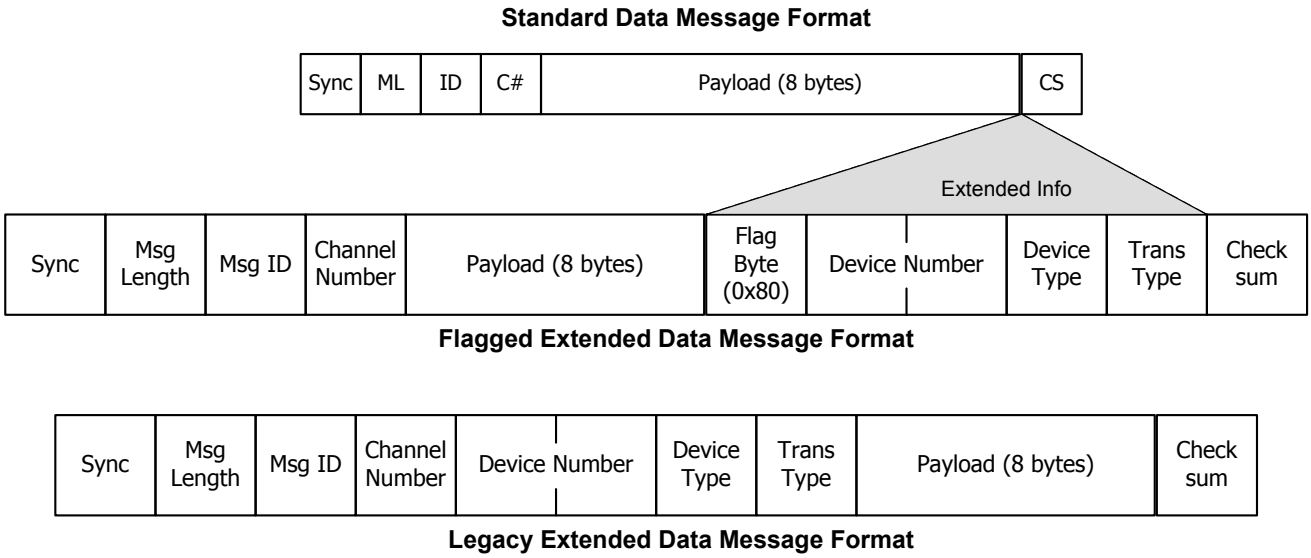


Figure 7-2. Extended Data Messages, Flagged and Legacy Formats.

Please note, the extended messaging format and Message ID are different for flagged and legacy extended messaging formats (refer to section 9.3 for message format).

There are two ways to configure extended messaging; Lib Config (0x6E) and Enable Extended Messages (0x66) (sections 9.5.2.19 and 9.5.2.16).Configuring extended messaging through Lib Config (0x6E) allows ANT to pass channel ID, RSSI, and timestamp information in the extended data bytes. Enable Extended Messages (0x66) allows for only the channel ID to be passed along with the received data message. Note that not all ANT devices support reception of extended messages. Refer to section 9.4 or request device capabilities (section 9.5.7.4).

If a burst sequence is received, only the first message in the sequence will contain extended data bytes.

7.1.1.1 Channel ID Output

ANT extended messaging allows for the transmitting device's channel ID (transmission type, device type and device number) to be passed to the host through the extended data bytes. If channel ID extended messaging is enabled, the flag byte will include 0x80 to indicate that channel ID information is present in the extended data bytes. An extended message that includes channel ID information will look like that in Figure 7-2. Channel ID Output is the only extended information available via the legacy extended format.

7.1.1.2 RSSI Output

Lib Config (0x6E) provides a way to receive the RSSI, or received signal strength indication, in addition to the 8-byte data payload. If RSSI extended messaging is enabled, the flag byte will include 0x40 to indicate that RSSI information is present in the extended data bytes. The received message including RSSI output is shown in Figure 7-3. For more information on RSSI extended messaging, refer to the "RSSI Extended Messaging" application note.

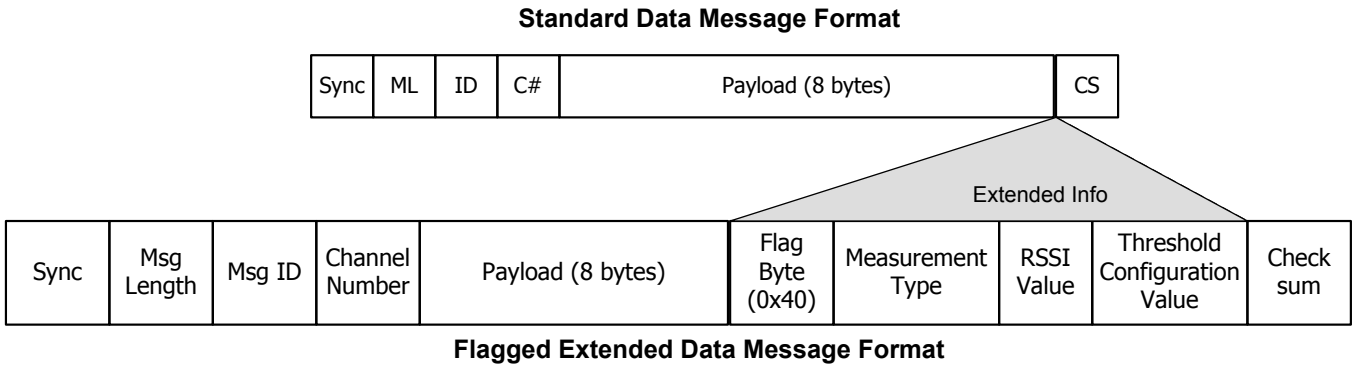


Figure 7-3. RSSI extended messaging

7.1.1.2.1 Measurement Type

The measurement value represents the measurement type of the received data message, and indicates how to interpret the RSSI Value field. The measurement type will be 0x20, which refers to DBM type. DBM type indicates that the RSSI value is taken in units of dBm. If the measurement type is any other value, do not decode any other bytes in the extended RSSI data. For more information refer to the "RSSI Extended Messaging" application note.

7.1.1.2.2 RSSI Value

The RSSI value is a signed integer that corresponds to the measured RSSI value in dBm.

7.1.1.2.3 Threshold Configuration Value

The threshold configuration value is used to indicate the dBm value of the bin configured using the Proximity Search command. The default value is -128 dB, which corresponds to an effective "Off" setting.

7.1.1.3 Timestamp Output

A device that supports the Lib Config command can also be configured to receive timestamp data along with the 8-byte data payload. A flag byte of 0x20 indicates that a device can expect timestamp information in the extended data bytes. The received extended message for timestamp output is show in Figure 7-4.

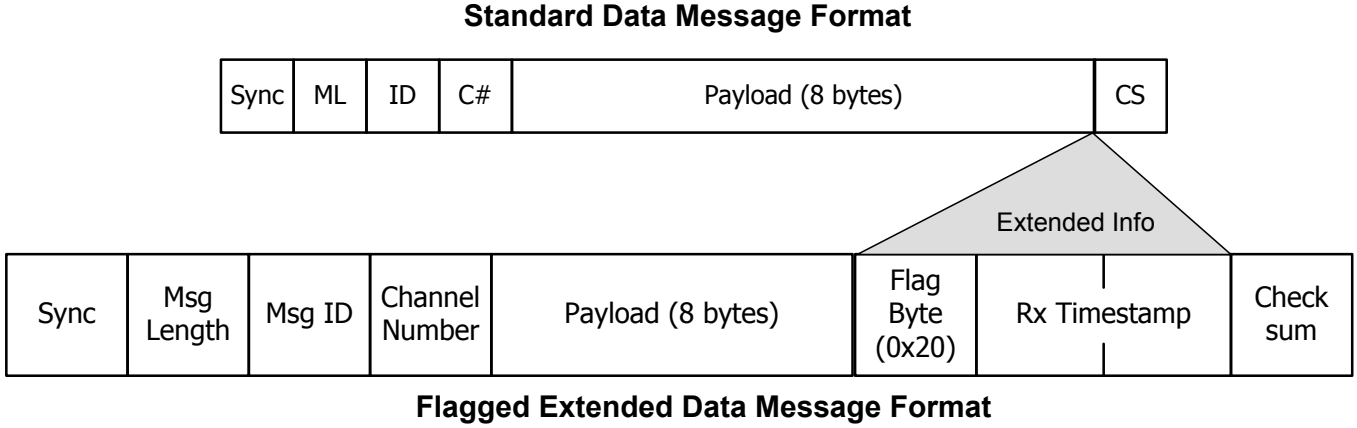


Figure 7-4. Timestamp extended messaging

7.1.1.3.1 Rx Timestamp

The Rx Timestamp is a 2 byte field (16-bit value) that rolls over every 2 seconds. This value is in little endian format. The timestamp is based on a 32 kHz clock.

7.1.1.4 Relative Order of Combined Extended Message Formats

One or more of the extended messaging flags may be implemented at one time (as long as Lib Config (0x6E) was used to configure extended messaging). If all three extended fields (channel ID, RSSI, and timestamp) are implemented the relative order of the extended data bytes will be channel ID, RSSI, and Rx timestamp, as shown in Figure 7-5.

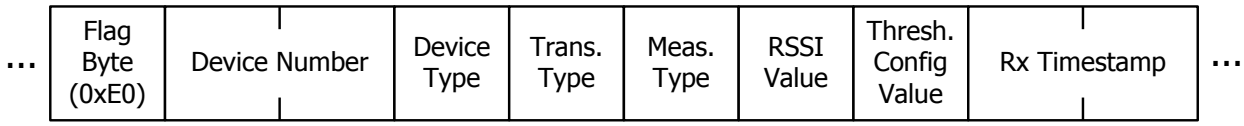


Figure 7-5. All fields enabled

If one or more of these elements are removed, the rightmost element moves left. For example, if channel ID and Rx timestamp are enabled, but RSSI is not, timestamp would shift to the left and is directly beside Channel ID, as shown in Figure 7-6

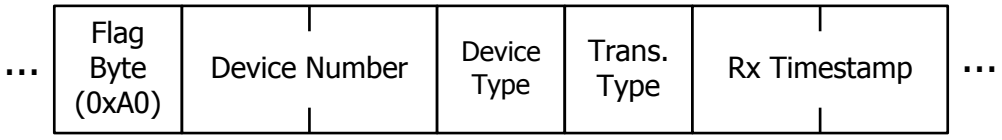


Figure 7-6. Channel ID and Rx timestamp enabled

Similarly, if RSSI and Rx timestamp are enabled, but channel ID is not, the received message will look like that in Figure 7-7.

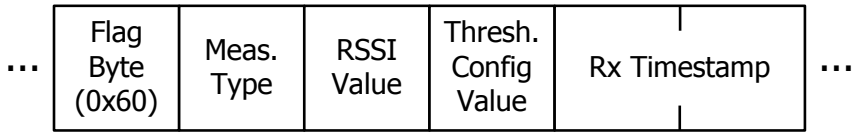


Figure 7-7. RSSI and Rx timestamp enabled

It is important to note that the message size will vary depending on the content of the extended data bytes.

7.2 Host MCU Serial Interface – Physical Layer

The ANT serial interface between host controller and ANT engine can be implemented over either a synchronous (SPI) or asynchronous (UART) connection. Unlike traditional SPI, the ANT serial connection uses four GPIO lines for control instead of a slave select; however, a standard SPI block is compatible with the ANT synchronous serial interface.

The connection type is selected by the product designer as preferred for the given implementation. The precise details of the physical and electrical interface of each ANT product can be found in each respective ANT product datasheet. Also refer to the Interfacing with ANT General Purpose Chipsets and Modules document for more details.

7.3 Host PC Serial Interface

The primary method of communication between ANT and a PC is through the ANT PC Interface Library. The components of this library are listed in section 9. Also refer to the "Dynamic Linking with ANT DLL" application note.

7.4 Interface to SoC

The details of the interface to each ANT product can be found in the product datasheet and SDK for the appropriate ANT product.

7.5 Mobile Devices Interface to ANT

Mobile apps for Android devices communicate with ANT hardware via the ANT Radio Service, an application available from the Google Play store. The ANT hardware in this case may be built natively into the mobile device and/or a dongle such as an ANT USB device. Refer to the "Creating ANT+ Android Applications" (within the Android ANT SDK package) document for more information.

8 Example ANT Network Implementation

A sample network implementation, presenting the features of the ANT protocol is shown in Figure 8-1 below.

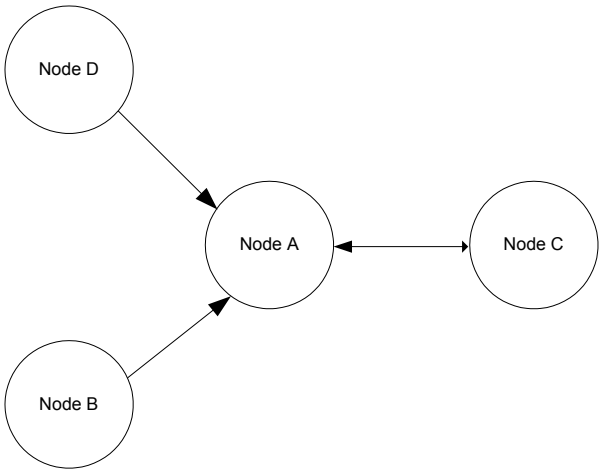


Figure 8-1. Example ANT network for implementation

The simple four-node network describes an application where information from multiple nodes (B, C and D) is to be received, and possibly analyzed, by a single central node (A). The arrows indicate the primary flow of information between the corresponding nodes. Note that nodes B, C and D only establish one channel, thus can be implemented using a single channel ANT device. Node A would require a 4 (or more) channel ANT device, as there are no 3 channel devices available on the market.

The following can be assumed:

- Node B uses the broadcast data type
- Node D uses the broadcast data type
- Node C requires the acknowledged data type
- All of the network prerequisites, such as network type, channel ID, RF frequency, etc. use default or known values between all nodes
- Device pairing has already been performed between the masters and their corresponding slaves

Sections 8.1 and 8.2 describe two methods of utilizing ANT to deploy the above example network.

8.1 Implementation using Independent Channels

Using independent channels is the simplest method of implementing the aforementioned network. Given the above assumptions, three independent channels are required. The configurations for the three independent channels are shown in the following tables.

Table 8-1. Channel between Node B and Node A where Node B will be the master

Node	Parameter	Value	Description
B	Network Number	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	1	Serial Number of Node B
	Transmission Type	1	Transmission Type (no shared address)
	Device Type	1	Device Type of Node B
	Channel Type	0x10	Bidirectional Transmit Channel
	Channel Period	8192	Default 4Hz Message Rate
	Data Type	0x4E	Broadcast
A	Network Number	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	1	Serial Number of Node B
	Transmission Type	1	Transmission Type (no shared address)
	Device Type	1	Device Type of Node B
	Channel Type	0x00	Bidirectional Receive Channel
	Channel Period	8192	Default 4Hz Message Rate
	Data Type	0x4E	Broadcast

Table 8-2. Channel between Node C and Node A where Node C will be the master

Node	Parameter	Value	Description
C	Network Number	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	10	Serial Number of Node C
	Transmission Type	1	Transmission Type (no shared address)
	Device Type	2	Device Type of Node C
	Channel Type	0x10	Bidirectional Transmit Channel
	Channel Period	8192	Default 4Hz Message Rate
	Data Type	0x4F	Acknowledged
A	Network Number	0	Default Public Network
	RF Frequency	66	Frequency 2466MHz
	Device Number	10	Serial Number of Node C
	Transmission Type	1	Transmission Type (no shared address)
	Device Type	2	Device Type of Node C

Node	Parameter	Value	Description
A	Channel Type	0x00	Bidirectional Receive Channel
	Channel Period	8192	Default 4Hz Message Rate
	Data Type	0x4F	Acknowledged

Table 8-3. Channel between Node D and Node A where Node D will be the master

Node	Parameter	Value	Description
D	Network Number	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	2	Serial Number of Node D
	Transmission Type	1	Transmission Type (no shared address)
	Device Type	1	Device Type of Node D
	Channel Type	0x10	Bidirectional Transmit Channel
	Channel Period	8192	Default 4Hz Message Rate
	Data Type	0x4E	Broadcast
A	Network Number	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	2	Serial Number of Node D
	Transmission Type	1	Transmission Type (no shared address)
	Device Type	1	Device Type of Node D
	Channel Type	0x00	Bidirectional Receive Channel
	Channel Period	8192	Default 4Hz Message Rate
	Data Type	0x4E	Broadcast

Section 8.1.1 details the sequence of events and message transactions between the host and ANT for each participating node as the above channels are established and network formed. Refer to section 5.3 for more information on the procedure for establishing a channel, and section 9 for more information regarding the various ANT commands.

8.1.1 Channel between Node B and Node A

The channel between Node B and Node A is established as shown in Figure 8-2.

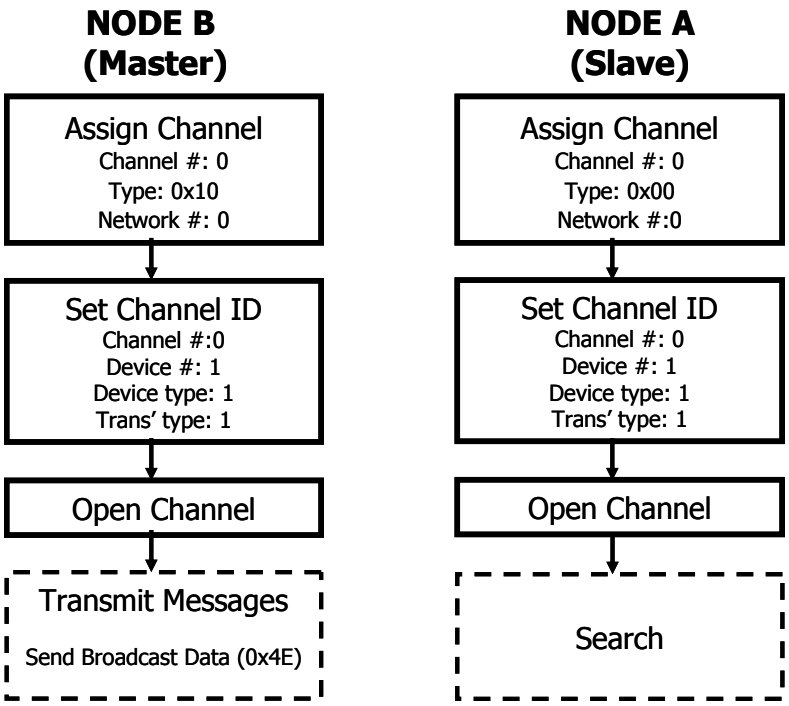


Figure 8-2. Node A & B Channel Establishment

As the network in this example uses the system defaults, only the minimum commands from host to ANT are required to establish the channel. The host issues the ANT_AssignChannel() and ANT_SetChannelId() messages with the configuration fields set as shown above. The channel number is assigned at the discretion of the host. In this case, it is channel zero for both; however, it should be noted that the **channel numbers do not need to match** on either side of the channel.

The host opens the channel using the ANT_OpenChannel() message. It is good practice to ensure the master channel is opened prior to the slave.

Once opened, the master's host provides ANT with data as it sees fit using the ANT_SendBroadcastData() message. Please note that the frequency at which the host provides ANT with new data may not be the same as the channel period. ANT will broadcast the data in its buffers at the desired message rate, if no new data is made available by the host, the previous data will be broadcast. However, appropriate safeguards to account for such repeated messages should be in place on the slave.

Once the slave's channel is opened, ANT will inform the host with a ChannelEventFunc() type message whenever a message from Node B is received. Based on the channel configuration settings, this will happen at 4Hz. If no message is received within the timeout period of the search, ANT will send the host a timeout message and close the channel.

8.1.2 Channel between Node C and Node A

The channel between Node B and Node A is established as shown in Figure 8-3.

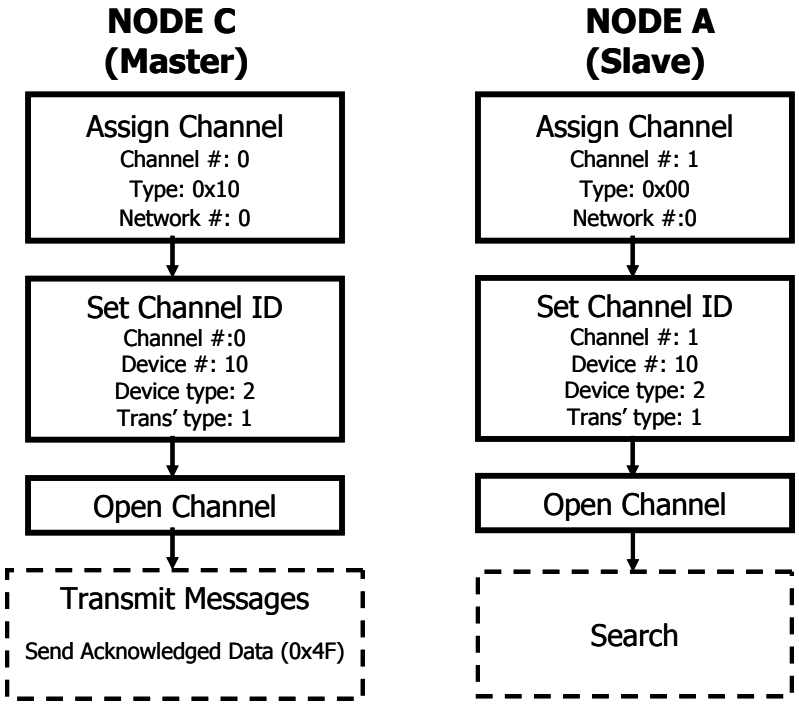


Figure 8-3. Node C & A Channel Establishment

The channel between nodes A and C are established as for nodes A and B above. Again, only the minimum commands from host to ANT are required to establish the channel with the given parameters. **Note, in this case the channel numbers do not match.** As Nodes B, C and D are single channel devices, their channel numbers will always be zero. Node A, on the other hand, is a 4 (or more) channel device and as such, will utilize channels 0, 1 and 2 in this example. As such, node A's channel 0 will be associated with Node B, channel 1 with Node C (as seen above) and channel 2 will be associated with node D as described in section 8.1.3.

Another difference in this channel, is that once the channel is opened, the master's host provides ANT with data as it sees fit using the ANT_SendAcknowledgedData() message. Also note, if no new data is made available by the host, the previous data will be sent as a broadcast message, not acknowledged message. This is the default message type as explained in section 5.4.1. Again, appropriate safeguards to account for such repeated messages should be in place on the slave. In this case, the slave could ignore any broadcast data types that are received from Node C, as all new data will be sent as acknowledged type and only repeated data will be of broadcast type.

Once the slave's channel is opened, ANT will inform the host with a ChannelEventFunc() type message whenever a message from Node C is received. Again, based on channel configuration settings, this will happen at 4Hz. If no message is received within the timeout period of the search, ANT will send the host a timeout message and close the channel.

8.1.3 Channel between Node D and Node A

The procedure for establishing the channel at Node D is exactly the same as that of Node B. The host of Node A will open a third channel to communicate with Node D in the same way as for Node B.

The independent channel network example that was implemented above will continue to function as it was deployed unless an application layer event dictates otherwise.

8.2 Implementation using Shared Channels

The network shown in Figure 8-1 can also be implemented as a single shared channel instead of using three independent channels. This would allow all nodes to be implemented using single channel ANT devices. The trade-off is increased power consumption (for the same latency) and, due to the inclusion of the shared address field, a reduction in the amount of maximum useful data; 8 bytes to 6 or 7 bytes per packet.

As mentioned in section 5.6, the central receiving node will be configured as master of the shared channel with the remaining nodes configured as its slaves. Each slave will have a unique one or two-byte shared channel address which shall be known only to it and the master. The updated network diagram for this setup is shown below.

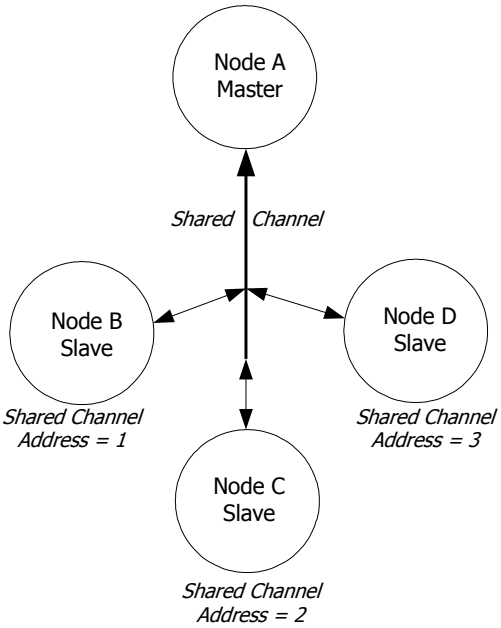


Figure 8-4. Shared channel implementation of sample network

Each node's channel configuration is shown in Table 8-4.

Table 8-4. Example shared channel node configuration.

Slave Node	Parameter	Value	Description
B	Network Type	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	3	Serial Number of Node A
	Transmission Type	3	Transmission Type (2 byte shared address)
	Device Type	3	Device Type of Node A
	Channel Type	0x20	Shared Receive Channel
	Channel Period	2370	~12Hz Message Rate
	Data Type	0x4E	Broadcast
C	Network Type	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	3	Serial Number of Node A
	Transmission Type	3	Transmission Type (2 byte shared address)
	Device Type	3	Device Type of Node A
	Channel Type	0x20	Shared Receive Channel
	Channel Period	2370	~12Hz Message Rate
	Data Type	0x4F	Acknowledged
D	Network Type	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	3	Serial Number of Node A
	Transmission Type	3	Transmission Type (2 byte shared address)
	Device Type	3	Device Type of Node A
	Channel Type	0x20	Shared Receive Channel
	Channel Period	2370	~12Hz Message Rate
	Data Type	0x4E	Broadcast
A	Network Type	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	3	Serial Number of Node A
	Transmission Type	3	Transmission Type (2 byte shared address)
	Device Type	3	Device Type of Node A
	Channel Type	0x30	Shared Transmit Channel
	Channel Period	2370	~12Hz Message Rate
	Data Type	0x4E	Broadcast

Please note:

The network type, RF frequency, device number, transmission type, device type and channel period are controlled by the master (Node A). All slaves that want to use this shared channel must adhere to these parameters.

The channel period for all nodes in the independent channel was 4Hz for each transmitting node (i.e. nodes B, C, D). In order to maintain this application-level channel period, each node in the shared channel actually needs to be set to a 12Hz channel period. This is the sum of the desired message rates of each slave node and will allow the master to service each node at a rate of 4Hz. For example, Node A may choose to cycle through the slaves, addressing Node B on the first channel period,

Node C on the next channel period, Node D on the next period, then back to Node B and so on. This will result in each node being addressed once every 4Hz. Similarly, the slaves will only be able to communicate back to the master at the time that they are serviced (i.e. also at 4Hz).

The channel between Node B and Node A is established as shown in Figure 8-5.

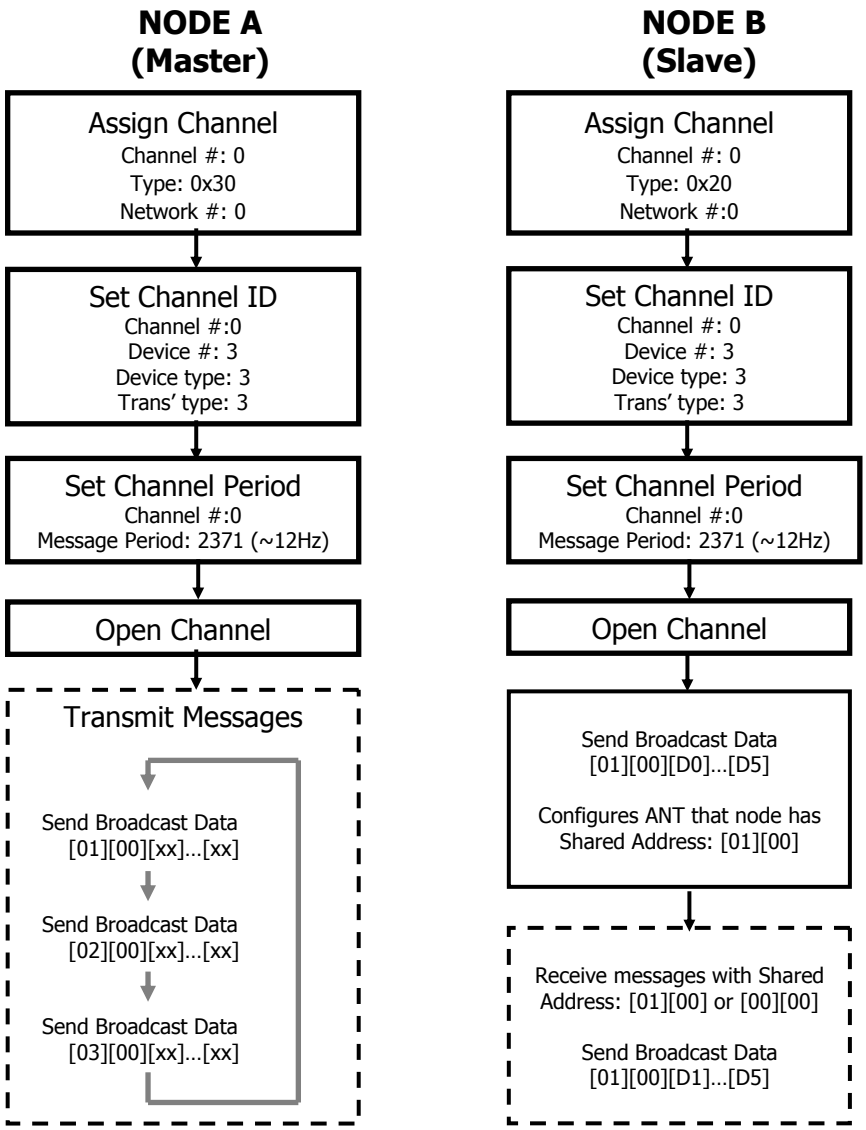


Figure 8-5. Shared Channel Example

Apart from the channel period, the network in this example uses the system defaults and only minimal commands from host to ANT are required to establish the channel. The host issues the ANT_AssignChannel(), ANT_SetChannelId() and the ANT_SetChannelPeriod() messages with the configuration fields set as shown above. The channel number is assigned at the discretion of the host. As all devices in this example are single channel, the channel number is zero for both; however, again, it should be noted that the **hosts' channel numbers do not need to match** on either side of the channel.

The host opens the channel using the ANT_OpenChannel() message. It is good practice to ensure the master channel is opened prior to any of the slaves. Once opened, the master's host should provide ANT with data on every channel period, using the ANT_SendBroadcastData() message. The host application should also pay special attention to the shared address

field, ensuring that the shared address field changes for each message sent. The shared address field should cycle through the shared addresses for Nodes B, C and D respectively, servicing each node at the desired 4Hz.

On the slave side, once the channel is opened, the host should send a single broadcast message to ANT with the first one or two bytes indicating Node B's shared channel address. This configures ANT to listen to messages that are addressed to Slave Node B. The host will now be informed each time ANT receives a message from the master that has Node B's shared channel address.

For this application, the slave's host would use the ANT_SendBroadcastData() message to provide data to ANT. ANT will send the data in the reverse direction whenever it receives the appropriately addressed message from the master (i.e. at 4Hz message rate).

Back on the master side, ANT will inform the host each time a message is received in the reverse direction from the slave with the corresponding shared channel address. For this particular network, each slave would send a message back to master Node A each time its own shared channel address appears. Slave nodes C and D are configured similarly to Node B as shown in Figure 8-6.

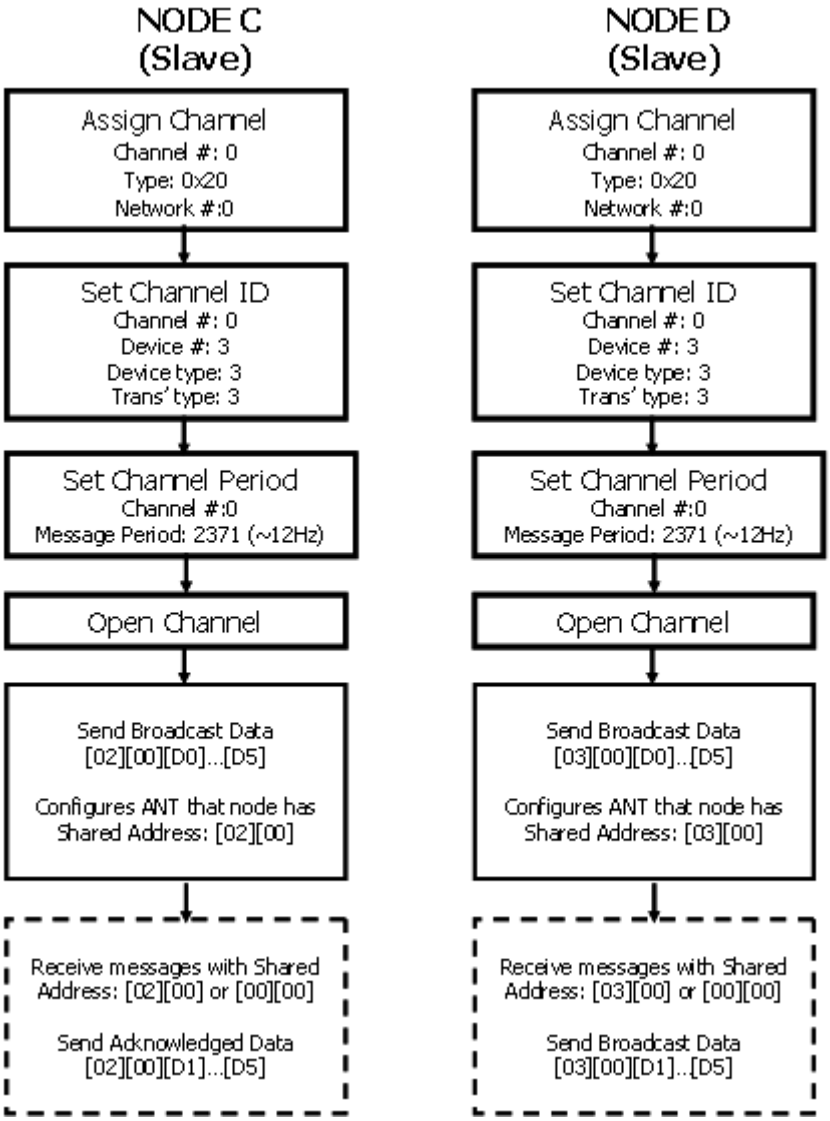


Figure 8-6. Slave Node C and D shared channel configuration

One difference being that the hosts send single ANT_SendBroadcastData() messages with the first one or two bytes changed to indicate the shared addresses of nodes C & D respectively. Each host will now be informed each time ANT receives a message from the master that has that node's shared channel address.

The only other difference, is that node C will use the ANT_SendAcknowledgedData() to provide data to ANT; which will then send to the master in the reverse direction whenever it receives the properly addressed message from the master (i.e. at 4Hz message rate).

The Independent and Shared Channel network implementations are to be used as a means for gaining familiarity with network design and deployment using ANT. The sample network could be implemented in other, more efficient ways, using various advanced features of ANT. In general, an application will govern the method of implementation that is best suited for its needs.

8.2.1 Shared Channel Transmission Type

The 2 least significant bits of the transmission type are used to determine the presence, and size, of a shared address field. This indicates that a shared channel can have transmission types from 0 to 3 only regardless of the number used. (i.e. trans. type of 6 (0110) is taken to be a 2 (10)).

For some ANT modules[1], the shared channel address will be a 2 byte field regardless of the number used for transmission type. However, newer ANT modules support the possibility of a 1 byte shared address and a 7 byte data payload.

In newer devices, a transmission type with the 2 least significant bits equal to 2 indicates a 1 byte shared address, as detailed in Table 5-2.

[1] AP2, AP2-USB, and CC257X modules support 1 or 2 byte shared addressing. AP1 and AT3 support 2 byte shared addressing only.

9 Appendix A – ANT Message Details

9.1 ANT Messages

A summary of the various messages that comprise the serial interface between ANT and a host is provided in section 9.3.

Note that all multi-byte fields are little endian in ANT messages. Additionally, all reserved bytes are set to zero unless otherwise stated.

9.1.1 Configuration Messages

The ANT configuration messages allow the Host to set or change various parameters of a channel, such as the network, device type, transmission type, message rate, RF frequency etc. These messages are the first step in enabling a system for ANT communication.

9.1.2 Notifications

Notifications allow ANT to inform the host of start-up conditions.

9.1.3 Control Messages

After desirable configuration of an ANT channel or channels, the control messages provide a method for supervising the RF as well as the activity of the ANT system.

9.1.4 Data Messages

The final step in establishing ANT communication; the data messages form the basic input and output of data from an ANT node. In a typical application, the Host will spend most of its ANT specific time on handling data messages.

9.1.5 Channel Event/Response Messages

The channel event/response messages are comprised of notifications and data that are sent from ANT to the Host. These include RF events that occur on a channel as well as messages that provide information about the state of the ANT system.

9.1.6 Requested Response Messages

The Host is able to obtain information from ANT using request messages. ANT replies to the requests using response messages. These include a summary of the capabilities, version information and status of channels.

9.1.7 Test Mode

ANT also accepts special test mode messages which allow the product developer or tester to verify the operation of the RF hardware by placing ANT in a RF continuous wave (CW) mode.

9.2 ANT Message Structure - Notes

The 'From' column in section 9.3 denotes the direction of data flow. An entry of 'ANT' indicates dataflow from ANT⇨Host. An entry of 'Host' indicates dataflow from Host⇨ANT.

The 'Reply' column in section 9.3 indicates whether ANT will send a response message to the respective command.

9.3 ANT Message Summary

Class	Type	Reply	From	Length	Msg ID	Message Content				
Config. Messages	Unassign Channel 9.5.2.1 page 63	Yes	Host	1	**0x41**	Channel Number				
	Assign Channel 9.5.2.2 page 63	Yes	Host	3	**0x42**	Channel Number	Channel Type	Network Number	[Extended Assign't]	
	Channel ID 9.5.2.3 page 65	Yes	Host	5	**0x51**	Channel Number	Device number (2 bytes)	Device Type ID	Trans. Type	
	Channel Period 9.5.2.4 page 66	Yes	Host	3	**0x43**	Channel Number	Channel Period (2 bytes)			
	Search Timeout 9.5.2.5 page 67	Yes	Host	2	**0x44**	Channel Number	Search Timeout			
	Channel RF Frequency 9.5.2.6 page 67	Yes	Host	2	**0x45**	Channel Number	RF Frequency			
	Set Network Key 9.5.2.7 page 68	Yes	Host	9	**0x46**	Network Number	Network Key (8 bytes)			
	Transmit Power 9.5.2.8 page 68	Yes	Host	2	**0x47**	0	TX Power			
	Add Channel ID to List 0 page 69	Yes	Host	6	**0x59**	Channel Number	Device number	Device Type ID	Trans. Type	List Index
	Add Encryption ID to List 9.5.2.10 page 69	Yes	Host	6	**0x59**	Channel Number	Encryption ID (4 bytes)	List Index		
	Config ID List 9.5.2.11 page 70	Yes	Host	3	**0x5A**	Channel Number	List Size	Exclude		
	Config Encryption ID List 9.5.2.12 page 71	Yes	Host	3	**0X5A**	Channel Number	List Size	List Type		

Class	Type	Reply	From	Length	Msg ID	Message Content			
Config. Messages	Set Channel Transmit Power 9.5.2.13 page 71	Yes	Host	2	**0x60**	Channel Number	Transmit Power		
	Low Priority Search Timeout 9.5.2.14 page 72	Yes	Host	2	**0x63**	Channel Number	Search Timeout		
	Serial Number Set Channel ID 9.5.2.15 page 73	Yes	Host	3	**0x65**	Channel Number	Device Type ID	Trans. Type	
	Enable Ext RX Messages 9.5.2.16 page 73	Yes	Host	2	**0x66**	0	Enable		
	Enable LED 9.5.2.17 page 74	Yes	Host	2	**0x68**	0	Enable		
	Crystal Enable 9.5.2.18 page 74	Yes	Host	1	**0x6D**	0			
	Lib Config 9.5.2.19 page 75	Yes	Host	2	**0x6E**	0	Lib Config		
	Frequency Agility 9.5.2.20 page 75	Yes	Host	4	**0x70**	Channel Number	Freq' 1	Freq' 2	Freq' 3
	Proximity Search 9.5.2.21 page 76	Yes	Host	2	**0x71**	Channel Number	Search Threshold		
	Configure Event Buffer 9.5.2.22 page 76	Yes	Host	6	**0x74**	0	Config	Size (2 bytes)	Time (2 bytes)
	Channel Search Priority 9.5.2.23 page 77	Yes	Host	2	**0x75**	Channel Number	Search Priority		
	High Duty Search 0 page 78	Yes	Host	2 or 3	**0x77**	Channel Number	Enable	Suppression Cycle (optional)	

Class	Type	Reply	From	Length	Msg ID	Message Content					
Config Messages	Configure Advanced Burst 9.5.2.25 page 79	Yes	Host	12	**0x78**	0	Enable	Max Packet Length	Required Features (3 bytes)	Optional Features (3 bytes)	Optional Stall Count / Retry Count Extension
	Configure Event Filter 9.5.2.26 page 80	Yes	Host	3	**0x79**	0	Event Filter (2 bytes)				
	Configure Selective Data Updates 9.5.2.27 page 81	Yes	Host	2	**0x7A**	Channel Number	Selected Data				
	Set Selective Data Update (SDU) Mask 0 page 82	Yes	Host	9	**0x7B**	SDU Mask Number	SDU Mask (8 bytes)				
	Configure User NVM 9.5.2.29 page 82	Yes	Host	Varies	**0x7C**	0	Address (2 bytes)	Data (variable length)			
	Enable Single Channel Encryption 9.5.2.30 page 83	Yes	Host	4	**0x7D**	Channel Number	Encryption Mode	Volatile Key Index	Decimation Rate		
	Set Encryption Key 9.5.2.31 page 84	Yes	Host	17	**0x7E**	Volatile Key Index	Encryption Key (16 bytes)				
	Set Encryption Info 9.5.2.32 page 84	Yes	Host	5, 20 or 17	**0x7F**	Set Parameter	Data String (4, 19, or 16 bytes)				
	Load/Store Encryption Key 9.5.2.33 page 85	Yes	Host	3 or 18	**0x83**	Operation	NVM Key Index	Volatile Key Index or Encryption Key			

Class	Type	Reply	From	Length	Msg ID	Message Content			
	Set USB Descriptor String 9.5.2.34 page 86	Yes	Host	Varies	**0xC7**	0	Descriptor String Number	Descriptor String - null terminated	String Length
Notifications	Start-up Message 9.5.3.1 page 87	N/A	ANT	1	**0x6F**	Start-up Message			
	Serial Error Message 9.5.3.2 page 87	N/A	ANT	1	**0xAE**	Error Number			
Control Messages	Reset System 9.5.4.1 page 88	No	Host	1	**0x4A**	0			
	Open Channel 9.5.4.2 page 88	Yes	Host	1	**0x4B**	Channel Number			
	Close Channel 9.5.4.3 page 88	Yes	Host	1	**0x4C**	Channel Number			
	Open Rx Scan Mode 9.5.4.5 page 89	Yes	Host	1	**0x5B**	0			
	Request Message 9.5.4.4 page 89	Yes	Host	Varies	**0x4D**	Channel Number/ Sub Message ID	Requested Message ID	Additional fields used with some requests	
	Sleep Message 9.5.4.6 page 90	No	Host	1	**0xC5**	0			
Data Messages	Broadcast Data 9.5.5.1 page 91	No	Host/ ANT	9	**0x4E**	Channel Number	Payload (8 bytes)		
	Acknowledged Data 9.5.5.2 page 94	No	Host/ ANT	9	**0x4F**	Channel Number	Payload (8 bytes)		
	Burst Transfer Data 9.5.5.3 page 98	No	Host/ ANT	9	**0x50**	Sequence/ Channel Number	Payload (8 bytes)		
	Advanced Burst Data 9.5.5.4 page 103	No	Host/ ANT	Varies	**0x72**	Sequence/ Channel Number	Payload (N bytes)		

Class	Type	Reply	From	Length	Msg ID	Message Content					
Channel Messages	Channel Event 9.5.6.1 page 108	N/A	ANT	3	**0x40**	Channel Number	1	Event Code	Optional Extended Event Parameters		
	Channel Response 9.5.6.1 page 108	N/A				Channel Number	Initiating Message ID	Response Code			
Requested Response Messages	Channel Status 9.5.7.1 page 113	N/A	ANT	2	**0x52**	Channel Number	Channel Status				
	Channel ID 0 page 114	N/A	ANT	5	**0x51**	Channel Number	Device number (2 bytes)	Device Type ID	Trans. Type		
	ANT Version 9.5.7.3 page 114	N/A	ANT	Varies	**0x3E**	Version (N bytes)					
	Capabilities 9.5.7.4 page 115	N/A	ANT	6	**0x54**	Max Channels	Max Networks	Standard Options	Advanced Options	Advanced Options 2	Advanced Options 3
	Serial Number 9.5.7.50 page 116	N/A	ANT	4	**0x61**	Serial Number (4 bytes)					
	Event Buffer Configuration 9.5.7.6 page 116	N/A	ANT	6	**0x74**	0	Buffer Config	Buffer Size (2 bytes)	Buffer Time (2 bytes)		
	Advanced Burst Capabilities 9.5.7.7 page 116	N/A	ANT	5	**0x78**	0	Supported Max Packet Length	Supported Features (3 bytes)			
	Advanced Burst Current Configuration 9.5.7.8 page 117			12		1	Enable	Max Packet Length	Required Features (3 bytes)	Optional Features (3 bytes)	Optional Stall Count / Retry Count Extension

Class	Type	Reply	From	Length	Msg ID	Message Content				
	Event Filter 9.5.7.9 page 118	N/A	ANT	3	**0x79**	0	Event Filter (2 bytes)			
	Selective Data Update Mask Setting 9.5.7.10 page 118	N/A	ANT	9	**0x7B**	SDU Mask Number	SDU Mask (8 bytes)			
	User NVM 9.5.7.11 page 118	N/A	ANT	Varies	**0x7C**	0	Data			
	Encryption Mode Parameters 9.5.7.12 page 119	N/A	ANT	2, 5, or 20	**0x7D**	Requested Encryption Parameter	Max Supported Mode, Encryption ID, or User Info String			
Test Mode	CW Init 9.5.8.1 page 120	Yes	Host	1	**0x53**	0				
	CW Test 9.5.8.2 page 120	Yes	Host	3	**0x48**	0	TX Power	RF Freq		
Extended Data Messages (Legacy)	Extended Broadcast Data¥ 9.5.9.1 page 121	No	Host/ ANT¥	13	**0x5D**	Channel Number	Device number (2 bytes)	Device Type ID	Trans. Type	Payload (8 bytes)
	Extended Acknowledged Data¥ 9.5.9.2 page 122	No	Host/ ANT¥	13	**0x5E**	Channel Number	Device number (2 bytes)	Device Type ID	Trans. Type	Payload (8 bytes)
	Extended Burst Data¥ 9.5.9.3 page 124	No	Host/ ANT¥	13	**0x5F**	Sequence/ Channel Number	Device number (2 bytes)	Device Type ID	Trans. Type	Payload (8 bytes)

¥ These are legacy formats to be used only with AT3 devices. Functions not supported by nRF24AP1 devices. nRF24AP2's devices only support these messages from Host -> ANT. For ANT->Host the additional bytes are appended to standard broadcast, acknowledged and burst data.

9.4 ANT Product Capabilities

9.4.1 Interface

Class	Type	nRF24AP1 and AP1 Modules	ANT11TRx 1 Chipsets & Modules	AT3 Chipsets & Modules	nRF24AP2 & AP2 Modules[2]	nRF24AP2 -USB	CC257x and C7 Modules[1]	USB Stick ANTUSB-m	nRF51 SOC
Config. Messages	Unassign Channel	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Assign Channel	Yes (3 bytes)	Yes (3 bytes)	Yes (3 or 4 bytes)	Yes (3 or 4 bytes)	Yes (3 or 4 bytes)	Yes (3 or 4 bytes)	Yes (3 or 4 bytes)	Yes (3 or 4 bytes)
	Channel ID	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Channel Period	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Search Timeout	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Channel RF Frequency	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Set Network Key	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Transmit Power	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Add Channel ID to List	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	Add Encryption ID to List	No	No	No	No	No	No	Yes	Yes
	Config ID List	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	Config Encryption ID List	No	No	No	No	No	No	Yes	Yes
	Set Channel Transmit Power	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	Low Priority Search Timeout	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	Serial Number Set Channel ID	No	No	Yes	No	No	Yes	Yes	No
	Enable Ext Rx Messages	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	Enable LED	No	No	Yes	No	No	No	No	No
	Crystal Enable	No	No	No	Yes	No	No	No	No
	Lib Config	No	No	No	Yes	Yes	Yes	Yes	Yes

[2] Note that this table reflects the current AP2 module's capabilities. Please refer to D00001363 Revision History – AP2 RF Transceiver Module for older modules.

Class	Type	nRF24AP1 and AP1 Modules	ANT11TRx 1 Chipsets & Modules	AT3 Chipsets & Modules	nRF24AP2 & AP2 Modules[2]	nRF24AP2 -USB	CC257x and C7 Modules[1]	USB Stick ANTUSB-m	nRF51 SOC
	Frequency Agility	No	No	No	Yes	Yes	Yes	Yes	Yes
	Proximity Search	No	No	No	Yes	Yes	Yes	Yes	Yes
	Configure Event Buffer	No	No	No	No	No	No	Yes	No
	Channel Search Priority	No	No	No	Yes	Yes	Yes	Yes	Yes
	High Duty Search	No	No	No	No	No	No	Yes	No
	Configure Advanced Burst	No	No	No	No	No	No	Yes	Yes
	Configure Event Filter	No	No	No	No	No	No	Yes	Yes
	Configure Selective Data Updates	No	No	No	No	No	No	Yes	Yes
	Set SDU Mask	No	No	No	No	No	No	Yes	Yes
	Configure User NVM	No	No	No	No	No	No	Yes	No
	Enable Single Channel Encryption	No	No	No	No	No	No	Yes	Yes
	Set Encryption Key	No	No	No	No	No	No	Yes	Yes
	Set Encryption Info	No	No	No	No	No	No	Yes	Yes
	Load/Store Encryption Key	No	No	No	No	No	No	Yes	No
	Set USB Descriptor String	No	No	No	No	Yes[3]	No	Yes	No
Notifications	Start up Message	No	No	No	Yes	Yes	Yes	Yes	Yes
	Serial Error Message	No	No	No	No	Yes	No	Yes	No
Control Messages	System Reset	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Open Channel	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Close Channel	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Open Rx Scan Mode	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	Request Message	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

[3] ANT USB2 does not support this command. It is only supported by nRF24AP2-USB.

Class	Type	nRF24AP1 and AP1 Modules	ANT11TRx 1 Chipsets & Modules	AT3 Chipsets & Modules	nRF24AP2 & AP2 Modules[2]	nRF24AP2 -USB	CC257x and C7 Modules[1]	USB Stick ANTUSB-m	nRF51 SOC
	Sleep Message	No	No	No	Yes	Yes	No	No	No
Data Messages Data Messages	Broadcast Data	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Acknowledge Data	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Burst Transfer Data	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Advanced Burst Data	No	No	No	No	No	No	Yes	Yes
Channel Event Messages	Channel Response / Event	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Requested Response Messages	Channel Status	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Channel ID	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	ANT Version	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Capabilities	Yes (4 bytes)	Yes (4 bytes)	Yes (6 bytes)	Yes (6 bytes)	Yes (6 bytes)	Yes (6 bytes)	Yes (7 bytes)	Yes (7 bytes)
	Serial Number	No	No	Yes	No	No	Yes	Yes	No
	Event Buffer Configuration	No	No	No	No	No	No	Yes	No
	Advanced Burst Capabilities	No	No	No	No	No	No	Yes	Yes
	Advanced Burst Current Configuration	No	No	No	No	No	No	Yes	Yes
	Event Filter	No	No	No	No	No	No	Yes	Yes
	SDU Mask Setting	No	No	No	No	No	No	Yes	Yes
	User NVM	No	No	No	No	No	No	Yes	No
	Encryption Mode Parameters	No	No	No	No	No	No	Yes	Yes
Test Mode	CW Init	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	CW Test	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

The CC257x and related modules also include extended serial messages to enable the Integrated ANT-FS feature. These are documented in the "D00001417 – Integrated FS/ANT-FS Interface Control Document".

9.4.2 Events

See section 9.5.6 for Event details.

Name	nRF24AP1 and AP1 Modules	ANT11TRx1 Chipsets and Modules	AT3 Chipsets and Modules	nRF24AP2 and AP2 Modules CC257x and C7 Modules[4]	nRF24AP2-USB	USB Stick ANTUSB-m	nRF51 SOC
RESPONSE_NO_ERROR	Yes	Yes	Yes	Yes	Yes	Yes	Yes
EVENT_RX_SEARCH_TIMEOUT	Yes	Yes	Yes	Yes	Yes	Yes	Yes
EVENT_RX_FAIL	Yes	Yes	Yes	Yes	Yes	Yes	Yes
EVENT_TX	Yes	Yes	Yes	Yes	Yes	Yes	Yes
EVENT_TRANSFER_RX_FAILED	Yes	Yes	Yes	Yes	Yes	Yes	Yes
EVENT_TRANSFER_TX_COMPLETED	Yes	Yes	Yes	Yes	Yes	Yes	Yes
EVENT_TRANSFER_TX_FAILED	Yes	Yes	Yes	Yes	Yes	Yes	Yes
EVENT_CHANNEL_CLOSED	Yes	Yes	Yes	Yes	Yes	Yes	Yes
EVENT_RX_FAIL_GO_TO_SEARCH	No	Yes	Yes	Yes	Yes	Yes	Yes
EVENT_CHANNEL_COLLISION	No	Yes	Yes	Yes	Yes	Yes	Yes
EVENT_TRANSFER_TX_START	No	No	Yes	Yes	Yes	Yes	Yes
EVENT_TRANSFER_NEXT_DATA_BLOCK	No	No	No	No	No	No	Yes
CHANNEL_IN_WRONG_STATE	Yes	Yes	Yes	Yes	Yes	Yes	Yes
CHANNEL_NOT_OPENED	Yes	Yes	Yes	Yes	Yes	Yes	Yes
CHANNEL_ID_NOT_SET	Yes	Yes	Yes	Yes	Yes	Yes	Yes
CLOSE_ALL_CHANNELS	No	No	Yes	Yes	Yes	Yes	Yes
TRANSFER_IN_PROGRESS	Yes	Yes	Yes	Yes	Yes	Yes	Yes
TRANSFER_SEQUENCE_NUMBER_ERROR	Yes	Yes	Yes	Yes	Yes	Yes	Yes
TRANSFER_IN_ERROR	No	No	Yes	Yes	Yes	Yes	Yes

[4] The CC257x and related modules also include extended events related to the Integrated ANT-FS feature. These are documented in the "D00001417 – Integrated FS/ANT-FS Interface Control Document".

Name	nRF24AP1 and AP1 Modules	ANT11TRx1 Chipsets and Modules	AT3 Chipsets and Modules	nRF24AP2 and AP2 Modules CC257x and C7 Modules[4]	nRF24AP2-USB	USB Stick ANTUSB-m	nRF51 SOC
TRANSFER_BUSY	No	No	No	No	No	No	Yes
MESSAGE_SIZE_EXCEEDS_LIMIT	No	No	No	Yes	Yes	Yes	Yes
INVALID_MESSAGE	Yes	Yes	Yes	Yes	Yes	Yes	Yes
INVALID_NETWORK_NUMBER	Yes	Yes	Yes	Yes	Yes	Yes	Yes
INVALID_LIST_ID	No	No	Yes	Yes	Yes	Yes	Yes
INVALID_SCAN_TX_CHANNEL	No	No	Yes	Yes	Yes	Yes	Yes
INVALID_PARAMETER_PROVIDED	No	No	No	Yes	Yes	Yes	Yes
EVENT_QUE_OVERFLOW	No	No	No	Yes	Yes	Yes	Yes
EVENT_SERIAL_QUE_OVERFLOW	No	No	No	No	Yes	Yes	No
NVM_FULL_ERROR	No	No	Yes	No	No	No	No
NVM_WRITE_ERROR	No	No	Yes	No	No	No	No
USB_STRING_WRITE_FAIL	No	No	No	No	Yes	Yes	No
MESG_SERIAL_ERROR_ID	No	No	No	No	Yes	Yes	No
ENCRYPT_NEGOTIATION_SUCCESS	No	No	No	No	No	Yes	Yes
ENCRYPT_NEGOTIATION_FAIL	No	No	No	No	No	Yes	Yes

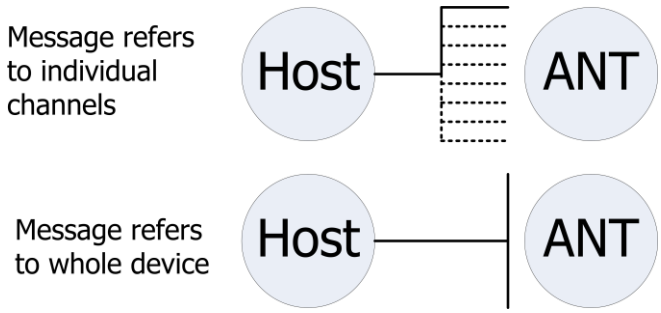
9.4.3 Output Power Level Settings

Different ANT modules support different output power level settings. The following table is a summary of the power level settings per ANT device.

Setting	nRF24AP1 and AP1 Modules	ANT11TRx1 Chipsets and Modules	AT3Chipsets and Modules,nRF24AP2, AP2 Modules and nRF24AP2-USB,and ANTUSB-m	nRF51xxx	CC257x and C7 Modules
0	-20 dBm	-20 dBm	-18 dBm	-20 dBm	-20 dBm
1	-10 dBm	-10 dBm	-12 dBm	-12 dBm	-10 dBm
2	-5 dBm	-5 dBm	-6 dBm	-4 dBm	-4 dBm
3	0 dBm	0 dBm	0 dBm	0 dBm	0 dBm
4	N/A	N/A	N/A	4dbm	4 dBm

9.5 ANT Message Details

This section provides detailed information regarding ANT messages and data fields for each ANT message type. The following symbols are used to indicate whether messages in this section refer to individual channels (e.g. Set Channel ID 0x51) or to the ANT device as a whole (e.g. Transmit Power 0x47):



9.5.1 ANT Constants

The constants vary depending on the selected ANT product (see product datasheet for further details):

1. MAX_CHAN – number of supported channels. Valid channels are 0..(MAX_CHAN-1).
2. MAX_NET – number of supported networks. Valid networks are 0..(MAX_NET-1).
3. MAX_BUFFER_SIZE – maximum number of bytes that can be stored before a buffer flush occurs
4. MAX_SDU_MASKS – number of supported SDU masks (selective data update)
5. MAX_ADDRESS – maximum address in the user NVM space.

These values can be determined for the specific ANT implementation by requesting the capabilities message (see section 9.5.7.4) or by referring to the device datasheet.

9.5.2 Configuration Messages

The following messages are used to configure a channel. Care should be taken to configure all appropriate pieces of information for a channel before opening it. All configuration commands return a response to indicate their success or failure. Therefore, a simple state machine can be setup for configuration of channels that advances states only when a RESPONSE_NO_ERROR is received for the current command and to re-send upon failures.

A simple timeout should also be implemented to protect against the case that a success/failure response is not received. Should this happen, the host should send ANT a series of 15 0's to effectively reset the ANT receive state machine. Please see the Interfacing with ANT General Purpose Chipsets and Modules Document for more information.

9.5.2.1 Unassign Channel (0x41)

BOOL ANT_UnAssignChannel(UCHAR ucChannel);

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN - 1	The channel to be unassigned.

```
// Example usage
ANT_AssignChannel(0, 0x00, 0);

ANT_UnAssignChannel(0);
```

This message is sent to the module to unassign a channel. A channel must be unassigned before it may be reassigned using the Assign Channel command.

9.5.2.2 Assign Channel (0x42)

BOOL ANT_AssignChannel(UCHAR ucChannel, UCHAR ucChannelType, UCHAR ucNetworkNumber);

OR

BOOL ANT_AssignChannelExt(UCHAR ucChannel, UCHAR ucChannelType, UCHAR ucNetworkNumber, UCHAR ucExtend);

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The channel number to be associated with the assigned channel. The channel number must be unique for every channel assigned on the module. The channel number must also be less than the maximum number of channels supported by the device.
Channel Type	UCHAR	As specified	Bidirectional Channels: 0x00 – Receive Channel 0x10 - Transmit Channel Unidirectional Channels: 0x50 – Transmit Only Channel 0x40 – Receive Only Channel Shared Channels: 0x20 – Shared Bidirectional Receive Channel 0x30 – Shared Bidirectional Transmit Channel

Parameters	Type	Range	Description
Network Number	UCHAR	0..MAX_NET-1	Specifies the network to be used for this channel. Set this to 0, to use the default public network. See section 5.2.5 for more details.
Extended Assignment [optional]	UCHAR	As specified	0x01 – Background Scanning Enable 0x04 – Frequency Agility Enable 0x10 – Fast Channel Initiation Enable 0x20 – Asynchronous Transmission Enable All other bits are reserved

```
// Example Usage
ANT_AssignChannel(0, 0x00, 0); // Receive channel 0 on network number 0; no extended assignment

OR

ANT_AssignChannelExt(0, 0x00, 0, 0x01); // Background scanning channel on channel 0, network number 0
```

This message is sent to ANT to assign a channel. Channel assignment reserves a channel number and assigns the type and network number to the channel. The optional extended assignment byte allows for the following features to be enabled: frequency agility, background scanning, fast channel initiation, and asynchronous transmission. For more information on these features see sections 5.2.1.4.1, 5.2.1.4.2, 5.2.1.4.4 and application notes "ANT Frequency Agility" and "ANT Channel Search and Background Scanning".

This Assign Channel command should be issued before any other channel configuration messages, and before the channel is opened. Assigning a channel sets all of the other configuration parameters to their defaults.

9.5.2.3 Set Channel ID (0x51)

BOOL ANT_SetChannelId(UCHAR ucChannel, USHORT usDeviceNum, UCHAR ucDeviceType, UCHAR ucTransmissionType);

Parameters	Type	Bit Range	Range	Description
Channel Number	UCHAR	-	0..MAX_CHAN-1	The channel number
Device Number	USHORT (little endian)	-	0..65535	The device number. For a slave, use 0 to match any device number.
Device Type MSB Pairing Request	UCHAR (1bit)	7	0..1	Pairing Request. Set this bit on master to request pairing Set this bit on slave to find a pairing transmitter.
Device Type bits 0:6 Device type ID	UCHAR (7bits)	0-6	0..127	The device type. For a slave use 0 to match any device type.
Transmission Type	UCHAR	-	0..255	The transmission type. For a slave use 0 to receive from any transmission type.

```
// Example Usage
// Tx channel
ANT_AssignChannel(0, 0x10, 0);
// wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 1234, 120, 1);
/*****************************************************/
// Rx channel
ANT_AssignChannel(0, 0x00, 0); // wildcard used for the device number
// wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 0, 120, 1); // device type 120 with pairing bit OFF
/*****************************************************/
// Pairing bit on Rx channel
ANT_AssignChannel(0, 0x00, 0); // wildcard used for the device number
// wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 0, 248, 1); // device type 120 with pairing bit ON
```

This message configures the channel ID for a specific channel.

The channel ID is intended to be unique (or nearly so) for each device link in a network. The ID is owned by the master. The master sets its ID, and the ID is transmitted along with its messages. The slave sets the channel ID to match the master it wishes to find. It may do this by providing the exact ID of the device it wishes to search for, or look for a class of device by setting a wildcard (0) for one of the subfields of the ID (Device Number, Device Type, or Transmission Type). When a match is found using a wildcard search, the Request Message command (with channel ID in its Message ID field) can be used to return the channel ID of the matched device. For more information refer to section 0.

If the device number is set to 0 on the slave, it will search for any masters that have matching device and transmission types. The state of the pair request bits must also match. This allows the product designer to choose the rules for pairing. If the designer wishes to pair two specific devices only when both sides agree, then the master and slave will both set the pairing bit when they wish to pair. If the designer intends for any slave of a certain type to pair to any master of a certain type, on a search at any time, then the pairing bit should always be set to 0.

When the device number is fully known the pairing bit is ignored i.e. if you know the exact device you are looking for, then pairing bit is irrelevant.

Note that transmission type and device type IDs are assigned and regulated to maintain network integrity and interoperability, except for the default public network. Please visit www.thisisant.com for more details on available standard network types or on how to obtain your own network type identifier.

9.5.2.4 Channel Messaging Period (0x43)

BOOL ANT_SetChannelPeriod(UCHAR ucChannel, USHORT usMessagePeriod);

Parameters	Type	Range	Default	Description
Channel Number	UCHAR	0..MAX_CHAN-1	-	The channel number
Messaging Period	USHORT (little endian)	0..65535	8192 (4Hz)	The channel messaging period in seconds * 32768. Maximum messaging period is ~2 seconds.

```
// Example Usage
ANT_AssignChannel(0, 0x00, 0); // receive channel on network number 0
// wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 0, 120, 123); // device number wild-card and pairing bit OFF
// wait for RESPONSE_NO_ERROR
ANT_SetChannelPeriod(0, 8192); // 4 Hz channel period
```

This message configures the messaging period of a specific channel where:

Messaging period = channel period time (s) * 32768.

E.g.: To send or receive a message at 4Hz, set the channel period to 32768/4 = 8192.

Note: The minimum acceptable channel period is difficult to specify as it is system dependent and depends on the number of configured channels and their use. Caution should be used to appropriately test the system when high data rates are used, especially in combination with multiple channels.

It is of critical importance that the channel period is defined in a manner consistent with the needs of the application. Some issues to consider are:

6. A smaller device period increases the message rate and thus increases system power consumption (see respective ANT product datasheet for details).
7. A smaller device period (faster message rate) allows higher Broadcast data-transfer rates.
8. A smaller device period (faster message rate) speeds up the device search operation.

Note: If the slave does not wish to receive data as fast as it is being transmitted, it may choose to receive data at a slower rate. This rate MUST be an integer divisor of the transmitted data rate, do not use non-integer divisors. For example, if the master is transmitting data at 4Hz (8192), the slave may prefer to receive data at 1Hz (32768). The slave will then receive 1 in 4 messages. This type of system provides the advantage of faster acquisition/reacquisition times due to the higher transmit data-rate, but maintains lower power consumption on the slave. Of course, the required data refresh rate on the slave needs to be considered if data messages are to be skipped.

9.5.2.5 Channel Search Timeout (0x44)

BOOL ANT_SetChannelSearchTimeout(UCHAR ucChannelNum, UCHAR ucSearchTimeout);

Parameters	Type	Range	Default	Description
Channel Number	UCHAR	0..MAX_CHAN-1	-	The channel number
Search Timeout	UCHAR	0..255	Non-AP1: 10 (25 seconds) AP1: 12 (30 seconds)	The search timeout to be used by this channel for receive searching. Each count in this parameter is equivalent to 2.5 seconds. i.e. 240 = 600 seconds = 10 minutes 0 - disable high priority search mode* 255 - infinite search timeout* *except for AP1: 0 = 0*2.5s = immediate timeout. 255 = 255*2.5 ~ 10.5mins

```
// Example Usage
ANT_AssignChannel(0, 0x00, 0); // receive channel on network number 0
// wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 0, 120, 123); // device number wild-card and pairing bit OFF
// wait for RESPONSE_NO_ERROR
ANT_SetChannelSearchTimeout(0, 24); // search timeout is 60s
```

This message is sent to the module to configure the length of time that the receiver will search for a channel before timing out. Note that a value of zero will disable high priority search mode, and a value of 255 sets an infinite search time-out. The exception to this is the AP1 module, which has only a high priority search mode. For AP1 only, a value of 0 is an immediate search timeout, and a value of 255 corresponds to approximately 10.5 minutes. For more information, refer to the "ANT Channel Search and Background Scanning Channel" application note.

9.5.2.6 Channel RF Frequency (0x45)

BOOL ANT_SetChannelRFFreq(UCHAR ucChannel, UCHAR ucRFFreq);

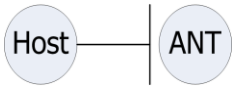
Parameters	Type	Range	Default	Description
Channel Number	UCHAR	0..MAX_CHAN-1	-	The channel to be assigned.
Channel RF Frequency	-UCHAR	0..124	66	Channel Frequency = 2400 MHz + Channel RF Frequency Number * 1.0 MHz

```
// Example Usage
ANT_AssignChannel(0, 0x10, 0); // transmit channel on network number 0
// wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 0, 120, 123); // device number wild-card and pairing bit OFF
// wait for RESPONSE_NO_ERROR
ANT_SetChannelRFFreq(0, 57); // RF frequency is 2457 MHz
```

This message is sent to ANT to set the RF frequency for a particular channel.

Great care should be taken in choosing an alternate value to the default. The selection of this channel may affect the ability to certify the product in certain global regions.

9.5.2.7 Set Network Key (0x46)



BOOL ANT_SetNetworkKey(UCHAR ucNetNumber, UCHAR *pucKey);

Parameters	Type	Range	Description
Network Number	UCHAR	0..MAX_NET-1	The network number
Network Key	UCHAR[8]	N/A	The 8 byte network key

```
// Example Usage
UCHAR aucNetworkKey = {0x00, 0x01, 0x00, 0x01, 0x00, 0x01, 0x00, 0x01}; // sample Network Key
ANT_SetNetworkKey(1, aucNetworkKey); // assign the network key to network number 1
// wait for RESPONSE_NO_ERROR
ANT_AssignChannel(0, 0x00, 1); // receive channel on network 1
```

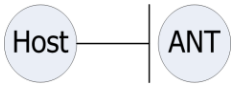
This message configures a network address for use by one of the available network numbers.

This command is not required when using the default public network. The default public network key is already assigned by default to Network Number 0. For nRF24AP1 devices, the remaining network numbers are left un-initialized. For most other ANT devices, all remaining network numbers default to the public network, with the exception of multi-mode chips. Refer to the device datasheet.

Only valid network keys will be accepted by ANT. Note, if a Set Network Key (0x46) command is sent with an invalid key, a RESPONSE_NO_ERROR may be received, but the network key will be unchanged; it will retain the value it held prior to the command being issued.

Note that network keys, transmission type, and device type IDs are assigned and regulated to maintain network integrity, and interoperability, except for the default public network. Please visit www.thisisant.com for more details on available standard network types or on how to obtain your own network key.

9.5.2.8 Transmit Power (0x47)



BOOL ANT_SetTransmitPower(UCHAR ucTransmitPower);

Parameters	Type	Range	Default	Description
Filler	UCHAR	0	0	A filler 0 byte that must be included
Transmit Power	UCHAR	0..4	3 (0dBm)	Refer to section 9.4.3

```
// Example Usage
ANT_SetTransmitPower(2); // set the RF output power to -5 dBm
```

This message is sent to the module to set the transmit power level for all channels.

This parameter must be used with extreme care. Setting the transmit power level to the highest level may not always be the most appropriate solution. Higher power levels increase current consumption, affect the sphere of influence for the device, and may have RF certification implications. A selected implementation must be tested to ensure that it meets the regulatory requirements of the region of intended sale.

9.5.2.9 Add Channel ID to List (0x59)

This message shares its message ID with the "Add Encryption ID to List" command. Please note this message is only used for slave channels. On ANT encrypted master channels the "Add Encryption ID to List" command behaviour described in 9.5.2.10 applies instead.

BOOL ANT_AddChannelID(UCHAR ucChannel, USHORT usDeviceNum, UCHAR ucDeviceType, UCHAR ucTransmissionType, UCHAR ucListIndex);

Parameters	Type	Bit Range	Range	Description
Channel Number	UCHAR	-	0..MAX_CHAN-1	The channel number
Device Number	USHORT (little endian)	-	0..65535	The device number.
Device Type ID	UCHAR (7bits)	0-6	0..127	The device type.
Transmission Type	UCHAR	-	0..255	The transmission type.
List Index	UCHAR	-	0..3	The index where the specified Channel ID is to be placed in the list.

```
// Example Usage
/*****************************************************/
// Rx channel
ANT_AssignChannel(0, 0x00, 0);
// wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 0, 120, 123); // device number is wild-card

ANT_AddChannelID(0, 145, 120, 123, 0); //add ID to list in index 0
ANT_AddChannelID(0, 152, 120, 123, 1); //add ID to list in index 1

ANT_ConfigList(0, 2, 0); //configure list as an inclusion list having 2 entries
ANT_OpenChannel(0);
```

Please note this message is only available on specific devices, refer to section 9.4, or request capabilities (section 9.5.7.4). This message is sent to the module to add channel IDs to the inclusion/exclusion list. When this list is used, these IDs will either be the only IDs accepted in a wild card search or IDs that will not be discovered at all. The use of these IDs is enabled by the Config List command detailed below. The inclusion/exclusion list is configured per channel. There is a maximum of 4 IDs allowed per list.

9.5.2.10 Add Encryption ID to List (0x59)

This message shares its message ID with the "Add Channel ID to List" command. Please note this message is only used for encrypted ANT master channels. On slave channels the "Add Channel ID" command behaviour described in 0 applies instead.

BOOL ANT_AddCryptoID(UCHAR ucChannel, UCHAR *pucCryptoID, UCHAR ucListIndex);

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The channel number
Encryption ID	UCHAR[4]	N/A	The unique 4 byte identifier of the negotiating slave
List Index	UCHAR	0..3	The index where the specified Encryption ID is to be placed in the list.

```
// Example Usage
// Encrypted Tx channel…
UCHAR aucCryptoId1 = {0x01, 0x23, 0x45, 0x67}; // example crypto ID
UCHAR aucCryptoId2 = {0x89, 0xAB, 0xCD, 0xEF}; // example crypto ID
ANT_AddCryptoID(0, aucCryptoId1, 0); // add ID1 to whitelist/blacklist in index 0 for channel 0
ANT_AddCryptoID(0, aucCryptoId2, 1); // add ID2 to whitelist/blacklist in index 1 for channel 0
```

Please note this message is only available on specific devices, refer to section 9.4, or request capabilities.

This message is sent to ANT to add encryption IDs to the whitelist/blacklist. When this list is used, these will either be the only encryption IDs allowed to negotiate with the encrypted master successfully, or these IDs will never be allowed to negotiate with the encrypted master successfully respectively. The use of these IDs is enabled by the ConfigCryptoList command detailed below. The whitelist/blacklist is configured per channel. There is a maximum of 4 IDs allowed per list.

9.5.2.11 Config ID List (0x5A)

BOOL ANT_ConfigList(UCHAR ucChannel, UCHAR ucListSize, UCHAR ucExclude);

Parameters	Type	Bit Range	Range	Description
Channel Number	UCHAR	-	0..MAX_CHAN-1	The channel number
List Size	UCHAR	-	0-4	The size of the inclusion/exclusion list
Exclude	UCHAR	-	0-1	Sets the list as include (0) or exclude (1)

```
// Example Usage
/*****************************************************/
// Rx channel
ANT_AssignChannel(0, 0x00, 0);
  // wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 0, 120, 123); // device number is wild-card

ANT_AddChannelID(0, 145, 120, 123, 0); //add ID to list in index 0
ANT_AddChannelID(0, 152, 120, 123, 1); //add ID to list in index 1

ANT_ConfigList(0, 2, 0); //configure list as an inclusion list having 2 entries
ANT_OpenChannel(0);
```

Please note this message is only available on specific devices, refer to section 9.4 or request device capabilities (section 9.5.7.4).

This message shares its message ID with the "Config Encryption List ID" command. Please note this message is only used for ANT slave channels. On encrypted ANT master channels the "Config Encryption List ID" command behaviour applies instead.

This message is sent to ANT to configure the inclusion/exclusion list The list size determines which channel IDs in the list are to be used; setting a size of 0 disables the inclusion/exclusion list, while setting a size of N includes the IDs stored at list indices 1-N (section 0). The exclude variable determines whether the IDs are to be found or to be ignored when the device is searching.

9.5.2.12 Config Encryption ID List (0x5A)

BOOL ANT_ConfigCryptoList(UCHAR ucChannel, UCHAR ucListSize, UCHAR ucListType);

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The channel number
List Size	UCHAR	0..4	The size of the encryption whitelist or blacklist
List Type	UCHAR	As specified	The whitelist will only allow slave encryption IDs on the list to negotiate successfully and the blacklist will never allow slave encryption IDs on the list to negotiate successfully: 0x00 – Whitelist 0x01 – Blacklist

```
// Example Usage
// Encrypted Tx channel…
UCHAR aucCryptoId1 = {0x01, 0x23, 0x45, 0x67}; // example crypto ID
UCHAR aucCryptoId2 = {0x89, 0xAB, 0xCD, 0xEF}; // example crypto ID
ANT_AddCryptoID(0, aucCryptoId1, 0); // add ID1 to crypto list in index 0 for channel 0
ANT_AddCryptoID(0, aucCryptoId2, 1); // add ID2 to crypto list in index 1 for channel 0
ANT_ConfigCryptoList(0, 2, 0x00); // configure crypto list for channel 0 as a whitelist having 2 entries
```

Please note this message is only available on specific devices, refer to section 9.4, or request capabilities.

This message shares its message ID with the "Config List ID" command. Please note this message is only used for encrypted ANT master channels. On slave channels the "Config List ID" command behaviour applies instead.

This message is sent to ANT to configure the whitelist/blacklist for the encrypted ANT master channel. The list size determines which encryption IDs in the list are to be used; setting a size of 0 disables the whitelist/blacklist, while setting a size of N includes the IDs stored at list indices 1-N (section 9.5.2.10). The list type variable determines whether the slave encryption IDs are to be whitelisted as the only devices allowed to negotiate, or blacklisted as the only devices immediately rejected by the encrypted ANT master channel.

9.5.2.13 Set Channel Tx Power (0x60)

BOOL ANT_SetChannelTxPower(UCHAR ucChannel, UCHAR ucTxPower);

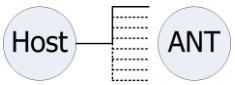
Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The channel number
Transmit Power	UCHAR	0..4	Refer to section 9.4.3.

```
// Example Usage
ANT_SetChannelTxPower(0, 3); // set the RF output power to 0 dBm on channel 0
```

This message is sent to the module to set the transmit power level for a specified channel. Please note this message is only available on specific devices, refer to section 9.4.

This parameter must be used with extreme care. Setting the transmit power level to the highest level may not always be the most appropriate solution. Higher power levels increase current consumption, affect the sphere of influence for the device, and may have RF certification implications. A selected implementation must be tested to ensure that it meets the regulatory requirements of the region of intended sale.

9.5.2.14 Channel Low Priority Search Timeout (0x63)



BOOL ANT_SetLowPriorityChannelSearchTimeout(UCHAR ucChannelNum, UCHAR ucSearchTimeout);

Parameters	Type	Range	Default	Description
Channel Number	UCHAR	0..MAX_CHAN-1	-	The channel number
Search Timeout	UCHAR	0..255	2 (5 seconds)	The search timeout to be used with by this channel for receive searching. Each count in this parameter is equivalent to 2.5 seconds. i.e. 240 = 600 seconds = 10 minutes A value of 0 will result in no low priority search. A value of 255 specifies infinite search time-out.

```
// Example Usage
ANT_AssignChannel(0, 0x00, 0); // receive channel on network number 0
// wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 0, 120, 123); // device number wild-card and pairing bit OFF
// wait for RESPONSE_NO_ERROR
ANT_SetLowPriorityChannelSearchTimeout(0, 24); // low priority search timeout is 60s
```

Please note this message is only available on specific devices, check datasheets for capabilities. This message is sent to ANT to configure the duration the receiver will search for a channel in low priority mode before switching to high priority mode. Unlike high priority mode, a low priority search will not interrupt other open channels on the device while searching. If the low-priority search times out, the module will switch to high priority mode until it either times out or the device is found. See the "ANT Channel Search and Background Scanning Channel" application note for more details.

9.5.2.15 Serial Number Set Channel ID (0x65)

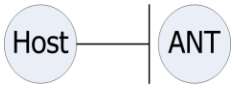
BOOL ANT_SetSerialNumChannelId(UCHAR ucChannel, UCHAR ucDeviceType, UCHAR ucTransmissionType);

Parameters	Type	Bit Range	Range	Description
Channel Number	UCHAR	-	0..MAX_CHAN-1	The channel number
Pairing Request	UCHAR (1bit)	7	0..1	Pairing Request. Set this bit on master to request pairing Set this bit on slave to find a pairing transmitter.
Device Type ID	UCHAR (7bits)	0-6	0..127	The device type. For a slave use 0 to match any device type.
Transmission Type	UCHAR	-	0..255	The transmission type. For a slave, use 0 to receive from any transmission type.

```
// Example Usage
// Tx channel
ANT_AssignChannel(0, 0x10, 0);
// wait for RESPONSE_NO_ERROR
ANT_SetSerialNumChannelId(0, 120, 123); // sets the lower 2 bytes of the serial number as the device number
/*****************************************************/
// Rx channel
ANT_AssignChannel(0, 0x00, 0); // wildcard used for the device number
// wait for RESPONSE_NO_ERROR
ANT_SetSerialNumChannelId(0, 120, 123); // device type 120 with pairing bit OFF
/*****************************************************/
// Pairing bit on Rx channel
ANT_AssignChannel(0, 0x00, 0); // wildcard used for the device number
// wait for RESPONSE_NO_ERROR
ANT_SetSerialNumChannelId(0, 248, 123); // device type 120 with pairing bit ON
```

Please note this message is only available on specific devices, refer to section 9.4 or request device capabilities (section 9.5.7.4). This message configures the channel ID to be used by a specific channel in the same way as the Channel ID command (refer to section 9.5.2.3); however it uses the two least significant bytes of the device's serial number as the device number.

9.5.2.16 Enable Extended Messages (0x66)



BOOL ANT_RxExtMesgsEnable (UCHAR ucEnable);

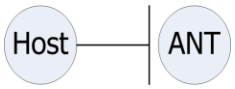
Parameters	Type	Range	Default	Description
Filler	UCHAR	0	0	A filler 0 byte that must be included
Enable	UCHAR	0..1	0	0 – Disable 1 – Enable

```
// Example Usage
ANT_RxExtMesgsEnable(1); // enable extended Rx messages
```

Please note this message is only available on specific devices, refer to section 9.4 or request device capabilities (section 9.5.7.4). This message is sent to ANT to enable or disable the extended Rx messages on the module. If supported, when

this setting is enabled ANT will include the channel ID with the data messages. Refer to section 7.1.1 for more information regarding extended data bytes.

9.5.2.17 Enable LED (0x68)



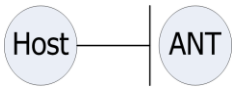
BOOL ANT_EnableLED(UCHAR ucEnable);

Parameters	Type	Range	Default	Description
Filler	UCHAR	0	0	A filler 0 byte that must be included
Enable	UCHAR	0..1	0	0 – Disable 1 – Enable

```
// Example Usage
ANT_EnableLED(1); // enable the LED
```

Please note this message is only available on specific devices, refer to section 9.4 or request device capabilities (section 9.5.7.4). This message is sent to the module to enable or disable the LED on the module. When the LED is enabled, it will blink each time an RF transmit or receive event is detected by the module.

9.5.2.18 Enable Crystal (0x6D)



BOOL ANT_CrystalEnable(void);

Parameters	Type	Range	Description
Enable	UCHAR	0	A filler 0 byte that must be included

```
// Example Usage
ANT_CrystalEnable(0); // enable an external 32kHz Crystal
```

Please note this message is only available on specific devices, check datasheets for capabilities. If the use of an external 32kHz crystal input is desired, this message must be sent once, each time a start-up message is received (described in section 9.5.3.1).

Enabling an external 32kHz crystal input as a low power clock source saves ~85uA while ANT is active when compared to using the internal clock source.

When using other external clock sources that are already available on the initialization, the Enable Crystal command is not required.

9.5.2.19 Lib Config (0x6E)

BOOL ANT_LibConfig(UCHAR ucLibConfig)

Parameters	Type	Range	Description
Filler	UCHAR	0	A filler 0 byte that must be included
Lib Config	UCHAR	As specified	0 – Disabled 0x20 – Enables Rx Timestamp Output 0x40 – Enables RSSI Output 0x80 – Enables Channel ID Output Other entries are combinations of set and cleared states

```
// Example Usage
ANT_LibConfib(0xE0); // enable Channel ID output, RSSI output, and Rx Timestamp output
```

Please note this message is only available on specific devices, refer to section 9.4. This message is sent to ANT to enable or disable the extended Rx messages on the module. If supported, when this setting is enabled ANT will include the channel ID, RSSI, or timestamp data with the data messages. See section 7.1.1 for more information regarding the extended data bytes.

9.5.2.20 Frequency Agility (0x70)

BOOL ANT_ConfigFrequencyAgility(UCHAR ucChannel, UCHAR ucFrequency1, UCHAR ucFrequency2, UCHAR ucFrequency3);

Parameters	Type	Range	Default	Description
Channel Number	UCHAR	0..MAX_CHAN-1	-	The channel number
ucFrequency1	UCHAR	0-124	3	Sets operating frequency 1 parameter for ANT frequency Agility.
ucFrequency2	UCHAR	0-124	39	Sets operating frequency 2 parameter for ANT frequency Agility.
ucFrequency3	UCHAR	0-124	75	Sets operating frequency 3 parameter for ANT frequency Agility.

```
// Example Usage
// Tx channel
ANT_AssignChannel(0, 0x10, 0, 0x04); //extended assignment byte enables frequency agility
  // wait for RESPONSE_NO_ERROR
ANT_ConfigFrequencyAgility(0, 5, 23, 80);
/*****************************************************/
// Rx channel
ANT_AssignChannel(0, 0x00, 0, 0x04); //extended assignment byte enables frequency agility
  // wait for RESPONSE_NO_ERROR
ANT_ConfigFrequencyAgility(0, 5, 23, 80); // Frequencies must match (in order)
/*****************************************************/
```

Please note this message is only available on specific devices, refer to section 9.4.1 or request device capabilities (section 9.5.7.4). This function configures the three operating RF frequencies for ANT frequency agility mode and should be used in

conjunction with the ANT_AssignChannel() extended byte (section 9.5.2.2). Should not be used with shared, or Tx/Rx only channel types. See section 5.2.1.4.1 and the "ANT Frequency Agility" application note for more details.

9.5.2.21 Proximity Search (0x71)

BOOL ANT_SetProximitySearch(UCHAR ucChannel, UCHAR ucSearchThreshold);

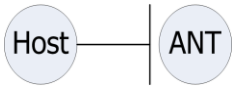
Parameters	Type	Range	Default	Description
Channel Number	UCHAR	0..MAX_CHAN-1	-	The channel number
ucSearchThreshold	UCHAR	0-10	0	Sets the proximity threshold bin: 0 – disabled 1:10 – closest to farthest

```
// Example Usage
// Rx channel
ANT_SetProximitySearch(0, 0x1); // search in nearest vicinity
```

Please note this message is only available on specific devices, refer to section 9.4 or request device capabilities (section 9.5.7.4). This function enables a one-time proximity requirement for searching. Only ANT devices within the set proximity bin can be acquired. Search threshold values are not correlated to specific distances as this will be dependent to the system design. A search threshold value of 1 (i.e. bin 1) will yield the smallest radius search and is generally recommended as there is less chance of connecting to the wrong device.

Once a proximity search has been successful, this threshold value will be cleared, effectively disabling the proximity search option. If another proximity search is desired, this command must be sent again prior to the next search. If the search times out, or if using a background scanning channel, the proximity threshold retains its value. Refer to the "Proximity Search" application note for more information.

9.5.2.22 Configure Event Buffer (0x74)



BOOL ANT_ConfigEventBuffer(UCHAR ucConfig,USHORT usSize, USHORT usTime)

Parameters	Type	Range	Description
Config	UCHAR	0x00 0x01	0x00 - Buffer Low Priority Events† 0x01 - Buffer all Events All other values are reserved
Size	USHORT	0-MAX_BUFFER_SIZE‡	Maximum number of bytes that will be stored before a buffer flush occurs Setting size to 0 disables event buffering
Time	USHORT	0-0xffff	Maximum time in 10ms units before a buffer flush occurs Setting time to 0 disables timer

```
// Example Usage
#define LOW_PRIORITY ((UCHAR)0x00);
#define ALL_EVENTS ((UCHAR)0x01);
USHORT usMyAppBufSize = 0x64;
USHORT usMaxBufTime = 0x03e8;

ANT_ConfigEventBuffer(LOW_PRIORITY, usMyAppBufSize, 0); // Buffer 100 bytes of events

ANT_ConfigEventBuffer(ALL_EVENTS, 0xffff, usMaxBufTime); // Buffer events for 10s at a time
```

†EVENT_TX, EVENT_RX_FAIL, and EVENT_CHANNEL COLLISION only
‡If the buffer size is set to a larger value than supported by the ANT device the device specific maximum is used instead

Please note this message is only available on specific devices, refer to section 9.4. This command may be sent to ANT to configure Event Buffering. Event Buffering allows the host to limit the frequency at which events are sent from the ANT device to the host. By deferring the processing of ANT Events, the host may remain in a lower power state for a longer period of time. Event Buffering may be used in conjunction with Event Filtering (Section 9.5.2.26).

Different groups of messages to buffer may be selected via the config argument (Low Priority Events or All Events)

The Buffer Size sets the number of bytes that can be stored before a flush occurs. Values that exceed the buffer size of the ANT device will instead be set to the maximum supported size. This maximum supported size can be determined by setting the buffer size to 0xffff and then requesting the EVENT_BUFFER_CONFIG message. Setting the buffer size to 0x00 disables buffering.

The Buffer Time is set in 10ms increments and sets the amount of time events are stored before a flush occurs. To use the Buffer Time, the Buffer Size must also be set to a non-zero value. Typically when using the Max Buffer Time the Buffer Size is set to the maximum value (0xffff). The buffer will be flushed on the next event after the buffer time elapses.

High priority events (Acknowledged or Burst messages) or other requests will trigger an event buffer flush before the buffer size or time expires.

9.5.2.23 Channel Search Priority (0x75)

BOOL ANT_SetChannelSearchPriority(UCHAR ucChannelNum, UCHAR ucSearchPriority)

Parameters	Type	Range	Default	Description
Channel Number	UCHAR	0..MAX_CHAN-1	-	The channel number
Search Priority	UCHAR	0…255	0	The search priority to be used for this channel. Higher numbers are given higher priority.

```
// Example Usage
ANT_AssignChannel(0, 0x00, 0); // receive channel on network number 0
//wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 0, 120, 123); //device number wild card and pairing bit OFF
//wait for RESPONSE_NO_ERROR
ANT_SetChannelSearchPriority(0, 2) //search priority is 2
```

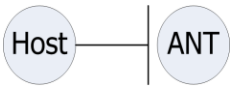
This message is used to configure the search priority of the channel. If a channel has a higher search priority, it will pre-empt lower search priority search channels that are already in progress. A pre-empted search will resume when the higher search priority search has either acquired a connection, or timed out. This functionality is primarily for determining precedence with multiple search channels that cannot co-exist (Search channels with different networks or RF frequency settings). Please note that this message is only available on specific devices, refer to section 9.4 for capabilities.

Example:

If both channel 0 and channel 1 have a search priority of 0, then whichever channel goes to search first is the search channel and the other channel must wait until it is finished before searching.

If channel 1 has a search priority of 2 and channel 0 has a search priority of 0, then channel 1 will become the search channel anytime it goes to search, forcing channel 0 to wait until channel 1 has either acquired a device, or times out.

9.5.2.24 High Duty Search (0x77)



BOOL ANT_ConfigHighDutySearch(UCHAR ucChannel, UCHAR ucEnable, UCHAR ucSuppressionCycle);

Parameters	Type	Range	Description
Filler	UCHAR	0	A filler 0 byte that must be included.
Enable	UCHAR	0..1	0x01: Enable High Duty Search 0x00: Disable High Duty Search
Suppression Cycle [optional]	UCHAR	0…5	Interval to allow high priority search in increments of 250ms. The search period is 1.25s. 0 enables high priority search full time and 5 suppresses it entirely. **The default setting is 3.** Not all parts support this setting.

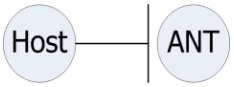
```
// Example Usage

// Rx channel
ANT_AssignChannel(0, 0x00, 0);

// wait for RESPONSE_NO_ERROR

ANT_SetChannelId(0, 0, 120, 123); // device number is wild-card

// wait for RESPONSE_NO_ERROR

ANT_ConfigHighDutySearch (0, 1, 4); // High duty search on channel 0, enable 1/1.25s low priority search

// wait for RESPONSE_NO_ERROR

ANT_OpenChannel(0);
```

This message is sent to enable high duty searching on a device. The command must be set with all channels closed, otherwise an error will be returned.

High duty search uses the entire available resources of the radio to search for a master device. The effect is that latency to acquire the master device is drastically reduced to an average of ½ period assuming ideal RF conditions. Once the device is acquired the channel becomes synchronized to the master. This mode of operation consumes high power while in search and should only be used in applications that have considerable power resources available such as PC and mobile applications.

A channel in high duty search can co-exist with other channels. However the effect of the high duty search on the other channels must be taken into account as a high priority searching scheme can interfere with the performance of other channels causing unacceptable outages due to constant channel collisions. To mitigate this problem the suppression cycle may be used to alternate windows of high and low priority search modes. A search window is defined to be 250ms within a period of 1.25s. The application has the option of setting X windows of low priority search within the period. For example, a suppression cycle of 1 opens up one 250ms low priority window within a 1.25s period. A suppression cycle of 0 uses high priority search across the entire period, whereas a suppression cycle of 5 uses low priority across the entire period. For more details please consult the "ANT Channel Search and Background Scanning" application note. Please note that not all parts that support high duty search also support the suppression cycle. For these parts the high duty search uses high priority mode at all times. High duty search may also be affected by co-existence with other protocols on multi-mode com chips. Please refer to section 9.4 and the datasheet of specific parts for more details.

9.5.2.25 Configure Advanced Burst (0x78)



BOOL ANT_ConfigAdvancedBurst(BOOL bEnable, UCHAR ucMaxPacketLength, ULONG ulRequiredFeatures, ULONG ulOptionalFeatures);

OR

BOOL ANT_ConfigAdvancedBurstExt(BOOL bEnable, UCHAR ucMaxPacketLength, ULONG ulRequiredFeatures, ULONG ulOptionalFeatures, USHORT usStallCount, UCHAR ucRetryCount);

Parameters	Type	Range	Description
Filler	UCHAR	0	A filler 0 byte that must be included
Enable	UCHAR	0..1	Enables/disables advanced burst mode. Advanced burst is disabled by default. 0x00 – Disable 0x01 – Enable
Max Packet Length	UCHAR	As specified	Specifies the maximum burst packet size that will be sent or received by the device: 0x01 – 8 byte 0x02 – 16 byte 0x03 – 24 byte
Required Features	ULONG (3 bytes, little endian)	As specified	Required advanced burst features. See "Supported Features" field in section 9.5.7.7 for a list of available features.
Optional Features	ULONG (3 bytes, little endian)	As specified	Optional advanced burst features. See "Supported Features" field in section 9.5.7.7 for a list of available features.
Stall Count [optional]	USHORT (little endian)	0..65535	Number of stall packets that will be sent before a transfer enters the retry count. Each packet corresponds to ~3ms. Typical default value is 3210 to provide ~10s of stalling. This byte is optional and does not need to be included
Retry Count Extension [optional]	UCHAR	0..255	Number of retry count cycle (5 retries) extensions. Typical default value is 4, providing 20 retries. This byte is optional and does not need to be included
// Example Usage ANT_ConfigAdvancedBurst(0, 0x01, 0x03, 0x01, 0x00); // Enable advanced burst, with maximum packet size of 24 bytes and frequency hopping as a required feature.			

This message is sent to ANT to enable and configure advanced burst. Advanced burst provides enhancements to the burst transfer mechanism, by increasing the transfer speed and providing additional features to improve the efficiency of transfers. For more details on available advanced burst features, please refer to the "Supported Features" field in section 9.5.7.7.

Advanced burst introduces support for burst packets of up to 24 bytes (transfer speed up to 60kbps). Devices with limited serial buffering may set a smaller maximum packet size; however, this will limit the transfer speed accordingly (up to 20 kbps for 8 byte packets, and 40 kbps for 16 byte packets). Note that the actual packet size used for the transfer will depend on the supported packet size on both ends.

Advanced burst features can be enabled independently as "required" or "optional". Required features not supported by the peer device will cause a transfer failure. Optional features not supported by the peer device will not cause a transfer failure and instead, will not be used for the transfer. Note that if particular advanced burst features are supported by an ANT device, but configured as disabled, they will be used in a burst transfer if the burst is initiated by another device requesting to use these features.

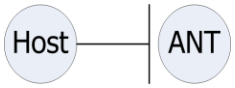
In order to make the burst transfer less susceptible to failure in the case of data throughput limitations between the host and ANT (i.e., if the host is unable to send data to ANT on a timely basis, or there is a lack of ability to push data to the host on the side receiving the burst), a stall mechanism is available. In this case, a configurable number of stall packets can be used to maintain the RF link and timing without transferring any data to the host. These packets are sent automatically by ANT and do not contain a data payload (similar to acknowledgements sent in response to acknowledged messages).

Additional retry cycles (one cycle = 5 retries) can be enabled to make transfers less susceptible to failure. This allows extending the number of retries for each burst packet beyond the default 5 retries used in normal burst transfers. Setting the Retry Count Extension requires the Stall Count to also be set. Retry counts should be synchronized between devices (i.e., they must match between master and slave). For interoperability with implementations from other manufacturers it is recommended to use the default value.

Advanced burst transfer is backwards compatible with devices that do not support this data type: if an advanced burst transfer is attempted with a device without this capability, the transfer will be downgraded to a normal burst transfer provided that a key property is not marked as required. This includes all advanced burst configuration options, including retry count extension and stall count.

Please note that this message is only available on specific devices, refer to section 9.4.1or request device capabilities (section 9.5.7.4).

9.5.2.26 Configure Event Filter (0x79)



BOOL ANT_ConfigEventFilter(USHORT usEventFilter)

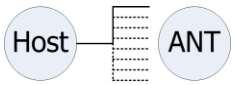
Parameters	Type	Range	Description
Filler	UCHAR	0	A filler 0 byte that must be included
Event Filter	USHORT (2 bytes, little endian)	0..65535	The Event Filter bit fields is as follows: Bit 0 – Filter event 1 (RESPONSE_NO_ERROR) Bit 1 – Filter event 2 (EVENT_RX_SEARCH_TIMEOUT) … Bit N – Filter event (N+1) where N is max 15 Setting a bit to 1 applies a filter to the corresponding event.

```
// Example Usage
ANT_ConfigEventFilter(0, 0x04); //Filter event value 3: EVENT_TX
```

Please note that this message is only available on specific devices as listed in section 9.4. This message is sent to ANT to apply a filter to specified events. Refer to section 9.5.6.1 for a list of events and their corresponding values.

Event filtering allows the host to prevent specific ANT events from being sent from the ANT device to the host. This allows the host to remain in a lower power state for longer, and to reduce processing. Event filtering may be used in conjunction with event buffering (section 9.5.2.22).This command will not yield significant power reduction when used on SOCs.

9.5.2.27 Configure Selective Data Updates (0x7A)



BOOL ANT_ConfigSelectiveDataUpdate(UCHAR ucChannel, UCHAR ucSelectedData)

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The channel number that SDU mask applies to.
Selected Data	UCHAR	N/A	The Selected Data bit fields is as follows: - Bits 0-4: SDU Mask Number (cannot exceed MAX_SDU_MASKS) - Bits 5-6: Reserved set to 0 - Bit 7: 0 -> Filter Broadcast messages only 1 -> Filter Broadcast and Acknowledged Messages A value of 0xFF disables the SDU Mask for the specified channel

```
// Example Usage
UCHAR aucSduMask = {0x00, 0x00, 0x00, 0x00, 0x00 ,0x00, 0x00, 0xFF }; // Configure mask to look for changes in the last byte of the payload
ANT_SetSduMask(0x01, aucSduMask); //Define SDU Mask 1
ANT_ConfigSelectiveDataUpdate(0x00, 0x81); //Apply SDU Mask 1 to broadcast and acknowledged messages on channel 0

//To clear the selective data updates setting from a channel:
ANT_ConfigSelectiveDataUpdate(0x00,0x00);
```

Please note that this message is only available on specific devices as listed in section 9.4. This message is sent to ANT to apply a selective data update (SDU) mask to a channel. Refer to section 0 for information about SDU masks.

Selective data updating allows an application to request that an ANT device only generates received data serial messages if data in specified bytes has changed. This could be useful for display units or similar devices which only need to update the display when the displayed data has actually changed and can remain in a lower power state otherwise.

This feature can be enabled for Broadcast messages only or Broadcast and Acknowledged messages. This feature does not apply to Burst Transfers. Once a channel is configured for selective data updates, it will remain configured until it is cleared (by resending the command with the selected data field set to 0xFF).

Note that if this feature is used in conjunction with single channel encryption, then the selective data update mask will be applied after the data has been decrypted.

9.5.2.28 Set Selective Data Update (SDU) Mask (0x7B)

BOOL ANT_SetSduMask(UCHAR ucSduMaskNumber, UCHAR *pucMask)

Parameters	Type	Range	Description
SDU Mask Number	UCHAR	0..MAX_SDU_MASKS	The SDU mask number to be associated with the SDU Mask
SDU Mask (8 bytes)	UCHAR[8]	N/A	Sets the bits in an 8-byte message that should be compared for selective data update purposes. 0 – Ignore 1 – Compare and send data update when this data changes

```
// Example Usage
UCHAR aucSduMask = {0x00, 0x00, 0x00, 0x00, 0x00 ,0x00, 0x00, 0xFF }; // Configure mask to look for changes in the last byte of the payload
ANT_SetSduMask(0x01, aucSduMask); //Define SDU Mask 1
```

Please note that this message is only available on specific devices as listed in section 9.4. This message is sent to ANT to define a selective data update mask.

Selective data update masks consist of an 8 byte bit field, where each bit indicates whether to ignore or process data in the corresponding bit locations in the payload of a received data message. For example, consider the following SDU mask:

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF

Setting this Mask value would indicate that the application should only be notified if any bit in the last byte in a standard message payload changes.

9.5.2.29 Configure User NVM (0x7C)

BOOL ANT_ConfigUserNVM(USHORT usAddr, UCHAR* ucData)

Parameters	Type	Range	Description
Filler	UCHAR	0	A filler 0 byte that must be included
Address	USHORT	0..MAX_ADDRESS¥	Little endian offset into user space where data will be written
Data	UCHAR[]		Variable length binary user data to be stored

```
// Example Usage
USHORT myBaseAddr = 0x100;
UCHAR myAppData = "ANT+ Alliance";

ANT_ConfigUserNVM(myBaseAddr, myAppData, sizeof(myAppData));
```

¥The combination of Addr and Size must not exceed the user NVM space of the device.

Please note this message is only available on specific devices, refer to section 9.4.

This command is sent to ANT to configure the user NVM space on an ANT device with application specific data. This configuration should not be done while channels are active. There are no word boundaries, data may be written to any address within the user space. Block size may be arbitrary up to the device specific maximum (refer to the device datasheet). Data may not be written beyond the user address range. It is not possible to secure NVM so existing data may

be overwritten. Data may be read from the user NVM using the requested response message to request message 0x7C as described in section 9.5.7.11.

9.5.2.30 Enable Single Channel Encryption (0x7D)

BOOL ANT_EncryptedChannelEnable(UCHAR ucChannel, UCHAR ucEnable, UCHAR ucKeyNum, UCHAR ucDecimationRate)

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The channel number
Encryption Mode	UCHAR	As specified	The "include user data" option will pass a user data field to the master with the encryption negotiation: 0x00 – Disable 0x01 – Enable 0x02 – Enable and Include User Information String
Volatile Key Index	UCHAR	0	Specifies which active encryption key (stored in volatile memory) to reference for this channel
Decimation Rate	UCHAR	1..255	Division of the master channel rate by the slave's tracking channel rate

```
// Example Usage
ANT_EncryptedChannelEnable(0, 0x01, 0, 4); // Encrypt channel 0, normal enable with 1 Hz Slave tracking 4Hz Master
```

This message is sent to ANT to enable encryption/decryption of an ANT channel using 128-bit AES-CTR. Advanced burst mode **MUST** be enabled on both the master and slave nodes before single channel encryption can be enabled; see "Configure Advanced Burst (0x78)" for more detail. Single channel encryption operates independently from all other advanced burst mode configuration parameters.

Enabling "Include User Information String" (section 9.5.2.32) will cause the ANT slave channel to pass a 19-byte user information string to the encrypted ANT master channel if the negotiation is successful. This feature does not need to be enabled on the encrypted ANT master channel.

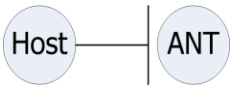
Single channel encryption can only be enabled on a slave channel once it has acquired an encrypted ANT master channel. Once it is enabled on the slave channel, the slave will automatically initiate a negotiation request with the encrypted ANT master channel node. ANT will respond with either an ENCRYPT_NEGOTIATION_SUCCESS or ENCRYPT_NEGOTIATION_FAIL event depending on the result of the attempted negotiation. If the negotiation is successful the user information string will be included with the ENCRYPT_NEGOTIATION_SUCCESS event. Refer to section 0 for details.

Single channel encryption must be re-enabled on a slave channel if the channel is closed or if the channel drops to search in order to trigger a new negotiation.

The decimation rate is the division of the encrypted ANT master channel period by the ANT slave channel period which is tracking it. The decimation rate parameter does not need to be specified on the ANT master and should be set to 1. For example, if an ANT slave channel tracking at 1 Hz was attempting to pair to an encrypted ANT master channel transmitting at 4 Hz, the decimation rate should be set to 4 on the ANT slave channel, and to 1 on the master channel.

Note that single channel encryption can only be enabled on one channel per node.

9.5.2.31 Set Encryption Key (0x7E)



BOOL ANT_SetCryptoKey(UCHAR ucKeyNum, UCHAR *pucKey);

Parameters	Type	Range	Description
Volatile Key Index	UCHAR	0	The reference index of the key in volatile memory to be set
Encryption Key	UCHAR[16]	N/A	The 128-bit secret key to be set

```
// Example Usage
UCHAR aucCryptoKey =
{0x03,0x01,0x04,0x01,0x05,0x09,0x02,0x06,0x05,0x03,0x05,0x08,0x09,0x07,0x09,0x03}; //example encryption key
ANT_SetCryptoKey(0, aucCryptoKey); // assign the example encryption key to key number 0
```

This message is sent to set the specified key location with a 16-byte encryption key which will be used for ANT single channel encryption/decryption. Note that only one volatile key index is available on current parts, and this parameter must therefore be set to zero.

9.5.2.32 Set Encryption Info (0x7F)

BOOL ANT_SetCryptoInfo(UCHAR ucSetParam, UCHAR *pucInfo);

Parameters	Type	Range	Description
Set Parameter	UCHAR	N/A	Determines the value being set by the string parameter: 0x00 – Encryption ID 0x01 – User Information String 0x02 – Random Number Seed
Data String	UCHAR[4] or UCHAR[19] or UCHAR[16]	N/A	The Encryption ID of the device to be set or the user information string to be set or the random number seed to be set depending on the set parameter operation requested respectively: Encryption ID - 4 Bytes User Information String - 19 Bytes Random Number Seed - 16 Bytes

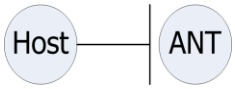
```
// Example Usage
UCHAR aucCryptoId = {0x00, 0x00, 0x04, 0x02}; // example encryption ID
ANT_SetCryptoInfo(0x00, aucCryptoId); // assign the example encryption ID
UCHAR aucUserInfo =
{0x03,0x01,0x04,0x01,0x05,0x09,0x02,0x06,0x05,0x03,0x05,0x08,0x09,0x07,0x09,0x03,0x02,0x03,0x08}; // example user
ANT_SetCryptoInfo(0x01, aucUserInfo); // assign the example user info string
UCHAR aucRanNumSeed =
{0x03,0x01,0x04,0x01,0x05,0x09,0x02,0x06,0x05,0x03,0x05,0x08,0x09,0x07,0x09,0x03}; // example random number seed
ANT_SetCryptoInfo(0x02, aucRanNumSeed); // assign the example random number seed
```

This message is used to set the parameters used in the ANT single channel encryption/decryption process:

- The encryption ID is a unique 4-byte string used for identification which is always passed with an encryption negotiation event from the master channel to the slave and vice versa. This value must be set each time a channel with single channel encryption enabled is assigned. The encryption ID is the parameter used to identify devices for encryption whitelist/blacklist purposes (section 9.5.2.10).
- The user information string is an optional 19-byte data string which is passed from the ANT slave to the encrypted ANT master channel after a successful negotiation event.

- The random number seed is required for certain devices which do not have a built-in random number generator and must be seeded by a cryptographically secure random number generator. Refer to the individual device's datasheet for information.

9.5.2.33 Load/Store Encryption Key from/in NVM (0x83)



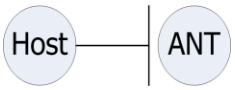
BOOL System_CryptoKeyNVMOp(UCHAR ucOperation, UCHAR *pucParams, UCHAR ucSize);

Parameters	Type	Range	Description
Operation	UCHAR	N/A	Determines the operation to be executed: 0x00 – Load Encryption Key from NVM 0x01 – Store Encryption Key in NVM
NVM Key Index	UCHAR	0..3	The index of the Encryption Key in NVM to be loaded or stored to depending on the selected operation
Volatile Key Index or Encryption Key	UCHAR or UCHAR[16]	0 or N/A	**When Operation is set to 0x00:** The index of the volatile key location that should be loaded with the NVM stored encryption key. **When Operation is set to 0x01:** The 128-bit Encryption Key to be stored to NVM

```
// Example Usage
System_CryptoKeyNVMOp(0x00, 1, 0); // copy encryption key stored in NVM key index 1 to volatile key index 0
UCHAR aucCryptoKey =
{0x03,0x01,0x04,0x01,0x05,0x09,0x02,0x06,0x05,0x03,0x05,0x08,0x09,0x07,0x09,0x03}; //example encryption key
System_ CryptoKeyNVMOp(0x01, 0, aucCryptoKey); // store the example encryption key in NVM key index 0
```

This message is used to load/store encryption keys from/to non volatile memory on the device. If a load operation is requested, the Volatile Key Index parameter is used and the command becomes 3 bytes long. If a store operation is requested the Encryption Key parameter is used and the command becomes 18 bytes long.

9.5.2.34 Set USB Descriptor String (0xC7)



BOOL ANT_SetUSBDescriptorString(UCHAR ucStringNum, UCHAR *pucDescString, UCHAR ucStringSize);

Parameters	Type	Range	Description
String Number	UCHAR	0..3	Descriptor String Number: 0 – PID/VID 1 – Manufacturer String 2 – Device String 3 – Serial Number String
String Character 0	UCHAR	0..255	String Character 0/VID LSB
String Character 1	UCHAR	0..255	String Character 1/VID MSB
String Character 2	UCHAR	0..255	String Character 2/PID LSB
String Character 3	UCHAR	0..255	String Character 3/PID MSB
String Character 4 – (N-1)	UCHAR	0..255	Remaining String Characters
String Character N	UCHAR	0	NULL character (except for string 0, PID/VID string)

```
// Example Usage
UCHAR aucDescString0 = {0xCF, 0x0F, 0x08, 0x10}; // sample VID/PID string
UCHAR aucDescString1 = "Dynastream Innovations"; // sample Manufacturer String
UCHAR aucDescString2 = "ANT USBStick2"; // sample Device String
UCHAR aucDescString3 = {'1', '2', '3', 0}; // sample Serial Number String (SN will be displayed by the OS as 123)

ANT_SetUSBDescriptorString (0, aucDescString0, sizeof(aucDescString0)); // set the VID/PID string
ANT_SetUSBDescriptorString (1, aucDescString1, sizeof(aucDescString1)); // set the Manufacturer String
ANT_SetUSBDescriptorString (2, aucDescString2, sizeof(aucDescString2)); // set the Device String
ANT_SetUSBDescriptorString (3, aucDescString3, sizeof(aucDescString3)); // set the Serial Number String
```

IMPORTANT: This message configures USB descriptor strings. **The AP2-USB does not support re-writeable flash memory**. Instead, space is allocated for three instances of each string descriptor. The last descriptor set is the one that is used. Once a descriptor has been set three times, it cannot be changed.

9.5.3 Notifications

9.5.3.1 Start-up Message(0x6F)

ResponseFunc (-, 0x6F)

Parameters	Type	Range	Description
Start-up Message	UCHAR	0..255	The Start-up Message bit field is as follows: 0x00 – POWER_ON_RESET Bit 0 – HARDWARE_RESET_LINE Bit 1 – WATCH_DOG_RESET Bit 5 – COMMAND_RESET Bit 6 – SYNCHRONOUS_RESET Bit 7 – SUSPEND_RESET Other bits are reserved

Please note this message is only available on specific devices, refer to section 9.4. The start-up message returns a 1-byte bit field, on every ANT power up or reset event. The bit field indicates the type of reset that occurred.

9.5.3.2 Serial Error Message (0xAE)

ResponseFunc(-,0xAE)

Parameters	Type	Range	Description
Serial Error Message	UCHAR	0..255	The first byte of data (usually the channel number) is the error number: Error number 0 – the first byte of the USB data packet was not the ANT serial message Tx sync byte (0xA4) Error number 2 – the checksum of the ANT message was incorrect Error number 3 – the size of the ANT message was too large The rest of the data contains a copy of the message that was sent.

Please note this message is only available on specific devices. Refer to section 9.4 for capabilities.

The Serial Error Message is sent in response to a poorly formed USB data. The data portion of this message may be used to debug the USB packet.

9.5.4 Control Messages

9.5.4.1 Reset System (0x4A)

BOOL ANT_ResetSystem(void);

Parameters	Type	Range	Description
Filler	UCHAR	0	

This message is sent to the module to reset the system and put it in a known, low-power state. Execution of this command terminates all channels. All information previously configured in the system can no longer be considered valid. After a Reset System command has been issued, the application should wait 500ms to ensure that ANT is in the proper, "after-reset" state before any further commands are issued from the host. For AT3 and newer modules, the RTS line can be monitored instead: only send commands after an RTS toggle has been observed. Please see the Interfacing with ANT General Purpose Chipsets and Modules Document for more information.

9.5.4.2 Open Channel (0x4B)

BOOL ANT_OpenChannel(UCHAR ucChannel);

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The number of the channel to be opened

This message is sent to the module to open a channel that has been previously assigned and configured with the configuration messages outline in prior sections. Execution of this command causes the channel to commence operation, and either data messages or events begin to be issued in association with this channel.

9.5.4.3 Close Channel (0x4C)

BOOL ANT_CloseChannel(UCHAR ucChannel);

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The number of the channel to be closed

This message is sent to close a channel that has been previously opened. The host will initially receive a RESPONSE_NO_ERROR message indicating the message was successfully received by ANT. The actual closing of the channel will be indicated by an EVENT_CHANNEL_CLOSED, and the host should wait for this message before performing any other operations on the channel.

When a channel is closed it remains assigned with all associated parameters remaining valid. The channel may be reopened at any time with the Open Channel Command.

9.5.4.4 Request Message (0x4D)

BOOL ANT_RequestMessage(UCHAR ucChannel, UCHAR ucMessageID);

OR

BOOL ANT_RequestMessage(UCHAR ucChannel, UCHAR ucMessageID, USHORT usAddr, UCHAR ucSize);

Parameters	Type	Range	Description
Channel Number/Sub Message ID	UCHAR	0..MAX_CHAN-1	The channel number associated with the message request, or sub message ID for commands that apply to the whole device. If requesting Advanced Burst Capabilities/Configuration, instead of channel number, set to: 0x00 – Request Advanced Burst Capabilities 0x01 – Request Advanced Burst Current Configuration
Message ID Requested	UCHAR	Refer to section 9.3	ID of the message being requested
addr¥	USHORT	Refer to section 9.5.7.11	Starting address to read, used when requesting User NVM message only
size¥	UCHAR	Refer to section 9.5.7.11	Block size to read, used when requesting User NVM message only

// Example Usage
ANT_RequestMessage(0, MESG_CHANNEL_ID_ID); // request the channel ID of channel 0
// response message contains the channel ID; no RESPONSE_NO_ERROR will be sent by ANT			

¥These arguments are only used when requesting the User NVM message. The combination of addr and size must not exceed the user NVM space of the device.

This message is sent to the device to request a specific information message from the device.

Valid messages include channel status, channel ID, ANT version, capabilities, event buffer, advanced burst capabilities/configuration, event filter, and user NVM. Requesting one of these messages causes ANT to send the appropriate response message. Please see these messages for specific details.

9.5.4.5 Open Rx Scan Mode (0x5B)

BOOL ANT_OpenRxScanMode();

Parameters	Type	Range	Description
Channel Number	UCHAR	0	Filler byte

// Example Usage
ANT_OpenRxScanMode();			

Please note this message is not available on all ANT modules; refer to section 9.4 or request the device capabilities (section 9.5.7.4).

This message is sent to the module in order to open a channel in continuous scan mode. The channel should have been previously assigned and configured as a slave receive channel. Execution of this command causes the channel to commence operation in continuous scanning mode. In this mode, the radio is active and receiving 100% of the time; no other channels may operate when the node is in continuous scanning mode. The node will pick up any message, regardless of the message period, that is transmitted on the correct RF frequency and matches its channel ID mask. It can receive from multiple devices simultaneously. It can also have messages pending to be sent to MAX_CHAN – 1 individual devices that are

communicating with the scanning device. This is achieved by passing an extended data message with the correct channel ID for the device the data is to be sent to on a channel in the range of 1:MAX_CHAN – 1. Refer to the "Continuous Scanning Mode for Asynchronous Topologies" application note.

9.5.4.6 Sleep Message (0xC5)

BOOL ANT_SleepMessage(void);

Parameters	Type	Range	Description
Filler	UCHAR	0	A filler 0 byte that must be included

```
// Example Usage
ANT_SleepMessage(0); // Puts ANT into deep sleep mode
```

Please note this message is only available on specific devices, refer to section 9.4. The Sleep command will put ANT into an ultra-low power mode (i.e. deep sleep_mode). Once this command has been issued, ANT will wait 1.2ms before attempting to enter this mode, by which time the SLEEP/(!MSGRDY) line must be set high. ANT will remain in this state until the SLEEP/(!MSGRDY) line is pulled low. Please refer to the "ANT Power States" application note and the "Interfacing with ANT General Purpose Chipsets and Modules" document for more details.

On exiting deep sleep mode, ANT will perform a reset and any prior configuration information will be lost.

9.5.5 Data Messages

There are four data types for sending and receiving data on a channel. These are described below.

9.5.5.1 Broadcast Data (0x4E)

BOOL ANT_SendBroadcastData(UCHAR ucChannel, UCHAR* pucBroadcastData); // Transmit

OR

ChannelEventFunc (Channel, EVENT_RX_BROADCAST) // Receive

On embedded platforms, the broadcast message may be processed the same as any other message received from ANT by processing the MESG_BROADCAST_DATA_ID (0x4E). In order to ensure appropriate message processing, check the message length field. For standard message packets, the message length will be 9. For flagged extended messages, the message length will be greater to account for the extra information appended to the data; check the flag byte for the presence of extended data bytes. The message length will be determined by the type of extended data present.

Note, data payload for a broadcast message is 8 bytes. The host application shall always define the full 8 bytes of the data packet and set unused bytes appropriately. If an application requires less than 8 bytes of data to be transmitted over the air, the remaining unused bytes shall be set to a predefined "unused" value (for example, ANT+ device profiles specify unused byte values, such as 0x00 or 0xFF).

For PC platforms, the ANT DLL will generate a channel event that may be processed the same as other events. The event is EVENT_RX_BROADCAST for standard broadcast messages and EVENT_RX_FLAG_BROADCAST for flagged extended data messages.

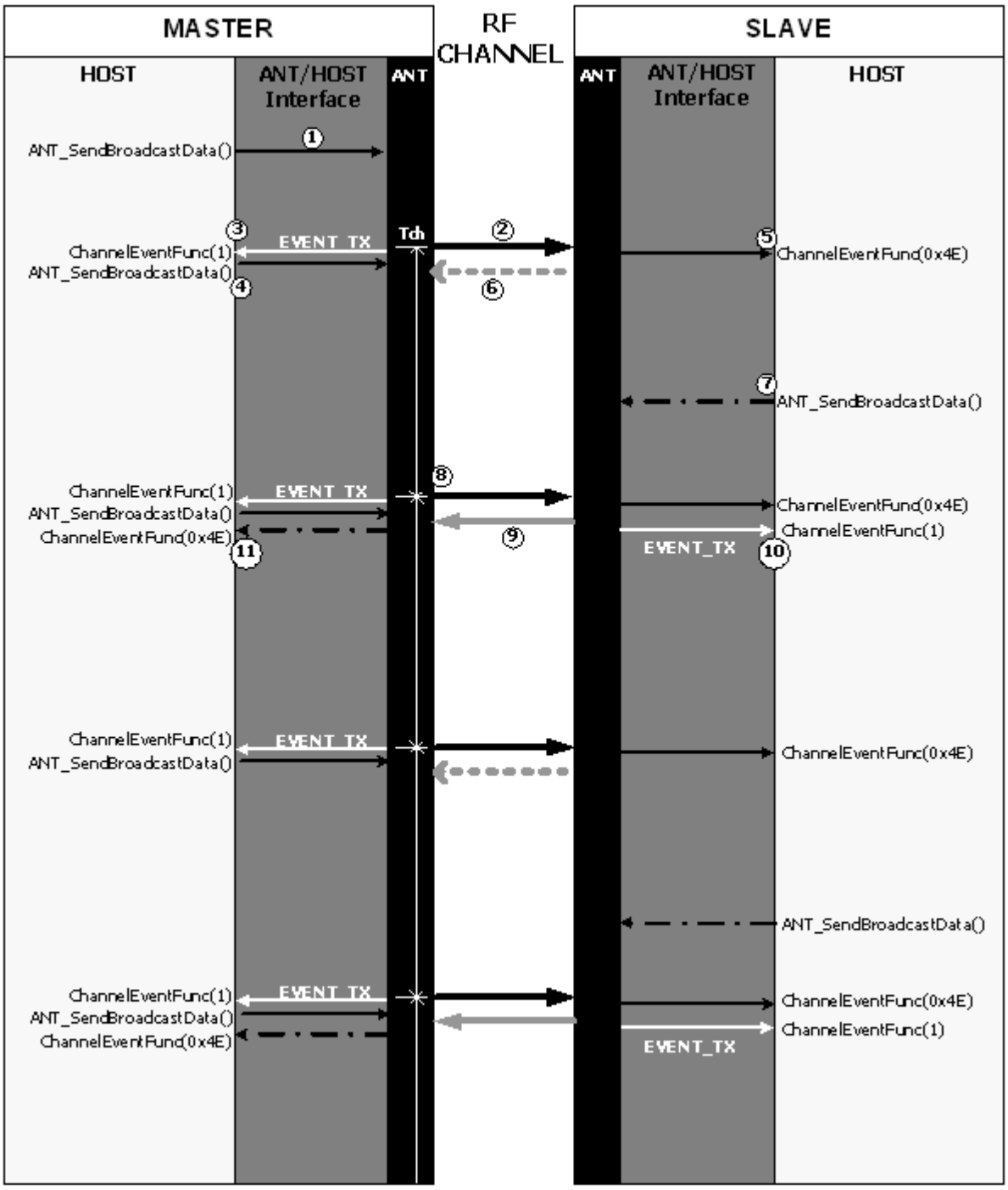
Please note that flagged extended data messages must be enabled using the ANT_RxExtMesgsEnable (0x66) or ANT_LibConfig(0x6E) messages.

Any application that processes flagged messages to get channel ID should also process legacy extended messages (MESG_EXT_BROADCAST_DATA_ID (0x5D) for embedded or EVENT_RX_EXT_BROADCAST for PC applications) to ensure compatibility.

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The channel the data is for/from
Data 0	UCHAR	0..255	The first data byte
..			
Data 7	UCHAR	0..255	The eighth data byte
[Flag Byte]	UCHAR	Refer to section 7.1.1	Indicates presence of extended data bytes
[Extended Data Bytes]	Varies, Refer to section 7.1.1	Refer to section 7.1.1	Optional extended messages bytes. Only included if flag byte indicates their presence. Refer to section 7.1.1 for more information on the possible extended data bytes.

```
// Example Usage
// Transmitter
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_TX:
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          ANT_SendBroadcastData(Channel_0, DATA);
          break;
        }
      }
      break;
    }
  }
}/*********************************************************/
// Receiver
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_RX_FLAG_BROADCAST:  // PC only; use MsgID 0x4E in embedded
    {
      UCHAR ucFlag = aucRxBuffer[9]; // First byte after the payload

      if(ucFlag & ANT_EXT_MESG_BITFIELD_DEVICE_ID)
      {
        // Channel ID of the device that we just received a message from.
        USHORT usDeviceNumber = aucRxBuffer [10] |( aucRxBuffer [11] << 8);
        UCHAR ucDeviceType =  aucRxBuffer [12];
        UCHAR ucTransmissionType = aucRxBuffer [13];
        printf("Chan ID(%d/%d/%d) - ", usDeviceNumber, ucDeviceType, ucTransmissionType);
      }
       // INTENTIONAL FALLTHROUGH
    }
    case EVENT_RX_BROADCAST:  // PC applications only; use MsgID 0x4E in embedded
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          // process received data which is in channel event buffer
          break;
        }
      }
      break;
    }
  }
}
```

Broadcast data is the default method of moving data between the transmitter and the receiver. Broadcast data is not acknowledged, therefore there is no way of knowing if it was actually received. Figure 9-1 below describes the broadcast message transactions from master host to ANT, over the RF channel to Slave ANT and host in the forward direction and



similarly in the reverse direction (Slave->Master).

Figure 9-1. Broadcast data sequence diagram

Master

A master ANT channel defaults to sending broadcast messages to the slave at the programmed channel period. The host uses an ANT_SendBroadcastData() message to send data to ANT (1), which will then buffer the data to be sent over the RF channel on the next designated time slot (i.e. channel period Tch). At the start of the next time slot, ANT sends the message over the RF channel (2) and issues the host an EVENT_TX Channel Event Function (3). This EVENT_TX message indicates to the host that ANT is ready to buffer new data. The host can send more data with another ANT_SendBroadcastData() command (4).

Once the slave's ANT receives the transmitted data, it will both notify and send data to the host with a ChannelEventFunc (0x4E) message (5). The slave has the option of sending data back in the reverse direction (6). In the case shown in Figure 9-1, the slave did not have any data to send, the dotted arrow is used to indicate the reverse direction, but no actual data sent.

On the next channel period (8), the process is repeated: ANT sends the data in its buffer over the RF channel, the master's host receives an EVENT_TX, and the slave's host receives the ChannelEventFunc (x04E). However, should the slave's host have requested a data transmission prior to that channel period (7), than it will be sent in the reverse direction on that timeslot (9). Similarly, an EVENT_TX ChannelEventFunc(1) will be sent from the slave's ANT to host (10) and a ChannelEventFunc (0x4E) from the master's slave will inform its host that a broadcast data type message was received (11).

The process above describes the message transactions for basic bidirectional broadcast operation.

Notes:

The EVENT_TX message can be used to prompt the master MCU that ANT is ready for the next data packet. It should NOT be used to prompt the slave MCU as, unlike the master, EVENT_TX does not necessarily occur on every channel period. This is illustrated in the example in Figure 9-1, where EVENT_TX occurs every second channel period. The ChannelEventFunc (0x4E), on the other hand, can be used instead as this does occur every channel period on the slave. These implementations are shown for both slave and master in the example usage at the beginning of this section.

If the slave does not manage to receive a data packet for its given time slot, an EVENT_RX_FAIL will be generated instead. No data is sent over the RF channel from slave to master on an EVENT_RX_FAIL.

If the host does not send the ANT_SendBroadcastData() message prior to the next channel timeslot, then the old data in ANT's buffer will be re-transmitted. It is up to the master's MCU to send new data on every message.

9.5.5.2 Acknowledged Data (0x4F)

BOOL ANT_SendAcknowledgedData(UCHAR ucChannel, UCHAR* pucBroadcastData); // Transmit

OR

ChannelEventFunc(Channel, EVENT_RX_ACKNOWLEDGED) // Receive

On embedded platforms, the acknowledged message may be processed as any other message received from ANT by processing the MESG_ACKNOWLEDGED_DATA_ID (0x4F). In order to ensure appropriate message processing, check the message length field. For standard message packets, the message length will be 9. For flagged extended messages, the message length will be greater to account for the extra information appended to the data; check the flag byte for the presence of extended data bytes.

Note that the data payload for an acknowledged message is 8 bytes. The host application shall always define the full 8 bytes of the data packet. If an application requires less than 8 bytes of data to be transmitted over the air, the remaining unused bytes shall be set to a predefined "unused" value (for example, ANT+ device profiles specify unused byte values, such as 0x00 or 0xFF).

For PC platforms, the ANT DLL will generate a channel event that may be processed in the same way as other events. The event is EVENT_RX_ACKNOWLEDGED for standard acknowledged messages and EVENT_RX_FLAG_ACKNOWLEDGED for flagged extended data messages.

Please note that flagged extended data messages must be enabled using the ANT_RxExtMesgsEnable (0x66) or ANT_LibConfig(0x6E) messages.

Any application that processes flagged extended messages to get channel ID should also process legacy extended messages (MESG_EXT_ACKNOWLEDGED_DATA_ID (0x5E) for embedded or EVENT_RX_EXT_ACKNOWLEDGED for PC applications) to ensure compatibility.

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The channel the data is for/from
Data 0	UCHAR	0..255	The first data byte
..			
Data 7	UCHAR	0..255	The eighth data byte
[Flag Byte]	UCHAR	Refer to section 7.1.1	Indicates presence of extended data bytes
[Extended Data Bytes]	Varies, Refer to section 7.1.1	Refer to section 7.1.1	Optional extended messages bytes. Only included if flag byte indicates their presence. Refer to section 7.1.1 for more information on the possible extended data bytes.

```
// Example Usage
// Transmitter
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_TRANSFER_TX_COMPLETED:
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          ANT_SendAckknowledgedData(Channel_0, DATA);
          break;
        }
      }
      break;

}
}
/*************************************************************/
// Receiver
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_RX_FLAG_ ACKNOWLEDGED:  // PC only; use MsgID 0x4E in embedded
    {
      UCHAR ucFlag = aucRxBuffer[9]; // First byte after the payload
```

```
      if(ucFlag & ANT_EXT_MESG_BITFIELD_DEVICE_ID)
      {
        // Channel ID of the device that we just received a message from.
        USHORT usDeviceNumber = aucRxBuffer [10] |( aucRxBuffer [11] << 8);
        UCHAR ucDeviceType =  aucRxBuffer [12];
        UCHAR ucTransmissionType = aucRxBuffer [13];

        printf("Chan ID(%d/%d/%d) - ", usDeviceNumber, ucDeviceType, ucTransmissionType);
      }
      // INTENTIONAL FALLTHROUGH
    }
    case EVENT_RX_ACKNOWLEDGED:  // PC only; use MsgID 0x4F in embedded
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          // process received data which is in channel event buffer
          break;
        }
      }
      break;
    }
  }
}
```

The Acknowledged Data message can be used in place of the Broadcast Data message to ensure the successful transmission of data. Acknowledged data is transmitted in the same transmission time slot as Broadcast Data but extends the length of the timeslot to accommodate the acknowledgement.

Acknowledged Data transmissions cannot be sent from channels configured for transmit only.

Acknowledged data messages can be sent from a slave to a master. In this case the message is sent immediately after a message is received from the master. If the acknowledged data message is sent successfully, this will be indicated by the channel event TRANSFER_TX_COMPLETED. If no acknowledgement is received from the master, this will be indicated by TX_TRANSFER_FAILED, which either means that the message failed to reach the master, or that the response from the master failed to reach the slave. In this case the slave's application layer may retry the message. A third possible scenario is that the channel is dropped and the slave begins to search; in this case the event GO_TO_SEARCH will be received, instead of either of the other two commands. If GO_TO_SEARCH is received then the acknowledged message will not be sent, and the channel should be unassigned

Figure 9-2 below describes the acknowledged message transactions from the master's host to ANT, over the RF channel to the slave ANT, host and vice versa in the reverse direction.

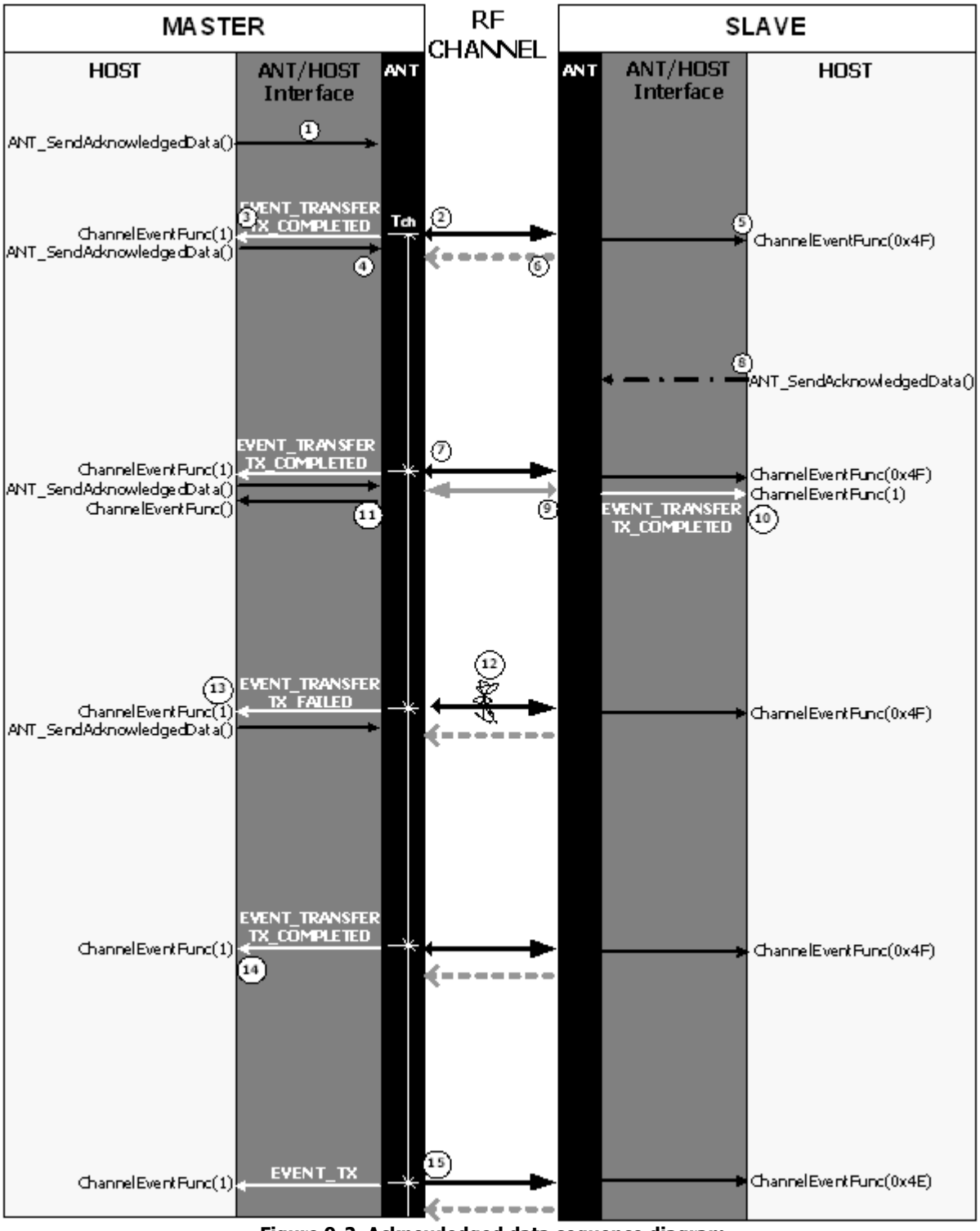


Figure 9-2. Acknowledged data sequence diagram

Similar to broadcast messaging, the host application requests the acknowledged data type when it sends the data payload to ANT with the ANT_SendAcknowledgedData() function (1); ANT buffers the data, which is transmitted on the next channel period (2). Unlike broadcast, the slave's ANT will automatically send an acknowledgement of receipt of data (this response indicated by the smaller arrowhead on (2)). If the master's ANT successfully receives this acknowledgement, it will send the host an EVENT_TRANSFER_TX_COMPLETED Channel Event Function (3). In this way, the master's host can be sure the message was transmitted successfully. Similar to broadcast and EVENT_TX, the EVENT_TRANSFER_TX_COMPLETED can be used to indicate to the host that ANT can receive new data. The host can send more data to ANT with another ANT_SendAcknowledgedData() command (4).

Once the slave's ANT receives the transmitted data, it will both notify and send data to the host with a ChannelEventFunc(0x4F) message (5). The slave has the option of sending data back in the reverse direction (6). In this case, the slave did not have any data to send, and the dotted arrow is used to indicate no actual data sent.

On the next channel period (7), the process repeats. However, should the slave's host have requested an acknowledged data transmission (8), this data will be sent in the reverse direction on that timeslot (9). The master's ANT will automatically send an acknowledgement of receipt (small arrowhead on (9)), and the slave's ANT, on receiving the acknowledgement, will send its host an EVENT_TRANSFER_TX_COMPLETED (10). The master's ANT will send a ChannelEventFunc(0x4F) both notifying and sending the data to its host (11).

Should the acknowledged message be subject to RF interference (12) and ANT fails to receive the appropriate acknowledgment, ANT will send an EVENT_TRANSFER_TX_FAILED to the host (13). This can occur for one of two reasons: either the recipient node (in this case the slave) never received the data and an acknowledgement was never sent; OR, the recipient (slave) got the data and sent an acknowledgement, but this failed to reach the originator (master).

Notes:

Similar to broadcast, the EVENT_TRANSFER_TX_COMPLETED or EVENT_TRANSFER_TX_FAILED can be used to indicate to the master MCU that ANT is ready for the next data packet. Also, on the slave side, the ChannelEventFunc (0x4F) function can be used to prompt the host for more data. These implementations are shown in the example usage at the beginning of this section.

If desired, the application can use EVENT_TRANSFER_TX_FAILED to resend the data. ANT does not automatically resend failed data.

Similar to broadcast, if the slave ANT fails to receive a message in the designated channel period, an EVENT_RX_FAIL occurs.

If the master host does not send any new data for the next channel timeslot (14 indicates the missing ANT_SendAcknowledgedData() command), then ANT will resend the old data as a broadcast message (15).

9.5.5.3 Burst Data (0x50)

```
BOOL ANT_SendBurstTransfer(UCHAR ucChannel, UCHAR* pucData, USHORT usNumDataPackets);

BOOL ANT_SendBurstTransferPacket(UCHAR ucChannelSeq, UCHAR* pucData);  // Transmit

        OR

ChannelEventFunc (Channel, EVENT_RX_BURST_PACKET)  // Receive
```

On embedded platforms, the burst message may be processed as any other message received from ANT by processing the MESG_BURST_DATA_ID (0x50). In order to ensure appropriate message processing, check the message length field. For standard message packets, the message length will be 9. For flagged extended messages, the first burst packet will have a message length greater than 9 to account for the extra information appended to the data; check the flag byte for the presence of extended data bytes. Subsequent message packets will not contain any extra messages and will be 9 bytes in length.

Note, data payload for a broadcast message is 8 bytes. The host application shall always define the full 8 bytes of the data packet and set unused bytes appropriately. If an application requires less than 8 bytes of data to be transmitted over the air, the remaining unused bytes shall be set to a predefined "unused" value (for example, ANT+ device profiles specify unused byte values, such as 0x00 or 0xFF).

For PC platforms, the ANT DLL will generate a channel event that may be processed the same as other events. The event is EVENT_RX_BURST for standard acknowledged messages and EVENT_RX_FLAG_BURST for flagged extended data messages. Note, for bursting only the first packet will contain the flag and extra information, the remaining burst packets will result in an EVENT_RX_BURST.

Please note that flagged extended data messages must be enabled using the ANT_RxExtMesgsEnable (0x66) or ANT_LibConfig (0x6E) messages.

Any application that processes flagged messages to get channel ID should also process legacy extended messages (MESG_EXT_BURST_DATA_ID (0x5F) for embedded or EVENT_RX_EXT_BURST for PC applications) to ensure compatibility.

Parameters	Type	Range	Description
Sequence Number	UCHAR (Bits 7:5)	As specified	The upper 3 bits of this byte are used as a sequence number to ensure transfer integrity (see below).
Channel Number	UCHAR (Bits 4:0)	0..MAX_CHAN-1	The lower 5 bits represent the channel number that the burst transfer is taking place on.
Data 0	UCHAR	0..255	The first data byte
..			
Data 7	UCHAR	0..255	The eighth data byte
[Flag Byte]	UCHAR	Refer to section 7.1.1	Indicates presence of extended data bytes
[Extended Data Bytes]	Varies, Refer to section 7.1.1	Refer to section 7.1.1	Optional extended messages bytes. Only included if flag byte indicates its presence. Refer to section 7.1.1 for more information on the possible extended data bytes.

```
// Example Usage
// Transmitter
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_TRANSFER_TX_COMPLETED:
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          ANT_SendBurstTransfer(Channel_0, DATA, 4); // 8 bytes per packet, 32 bytes total
          break;
        }
      }
      break;
    }
  }
}
/*************************************************************/
// Receiver
```

```
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_RX_FLAG_ BURST_PACKET:  // PC only; use MsgID 0x4E in embedded
    {
      UCHAR ucFlag = aucRxBuffer[9]; // First byte after the payload

      if(ucFlag & ANT_EXT_MESG_BITFIELD_DEVICE_ID)
      {
        // Channel ID of the device that we just received a message from.
        USHORT usDeviceNumber = aucRxBuffer [10] |( aucRxBuffer [11] << 8);
        UCHAR ucDeviceType =  aucRxBuffer [12];
        UCHAR ucTransmissionType = aucRxBuffer [13];

        printf("Chan ID(%d/%d/%d) - ", usDeviceNumber, ucDeviceType, ucTransmissionType);
      }
      // INTENTIONAL FALLTHROUGH
    }
    case EVENT_RX_BURST_PACKET:  // PC applications only; use MsgID 0x50 in embedded
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          // process received data which is in channel event buffer one packet at a time validating the
          // sequence
          break;
        }
      }
      break;
    }
  }
}
```

Burst data transmission is used to send larger amounts of data by sending messages continuously at the fastest rate possible. Each message packet in a Burst Transfer is acknowledged, and all lost packets are tried up to a maximum of 5 times to guarantee reception of the entire data transfer. Should a packet also fail on the 5th retry, the rest of the transfer will be aborted and ANT will send an error message to the host's MCU.

Transmission begins at the start of the normal time slot and multiple data packets are sent consecutively, extending the time slot for the duration of the burst transfer. Figure 9-3 below describes the burst message transactions from the master's host to ANT, over the RF channel to the slave's ANT, to host and back. Also refer to the application note "Burst Transfers" for more details.

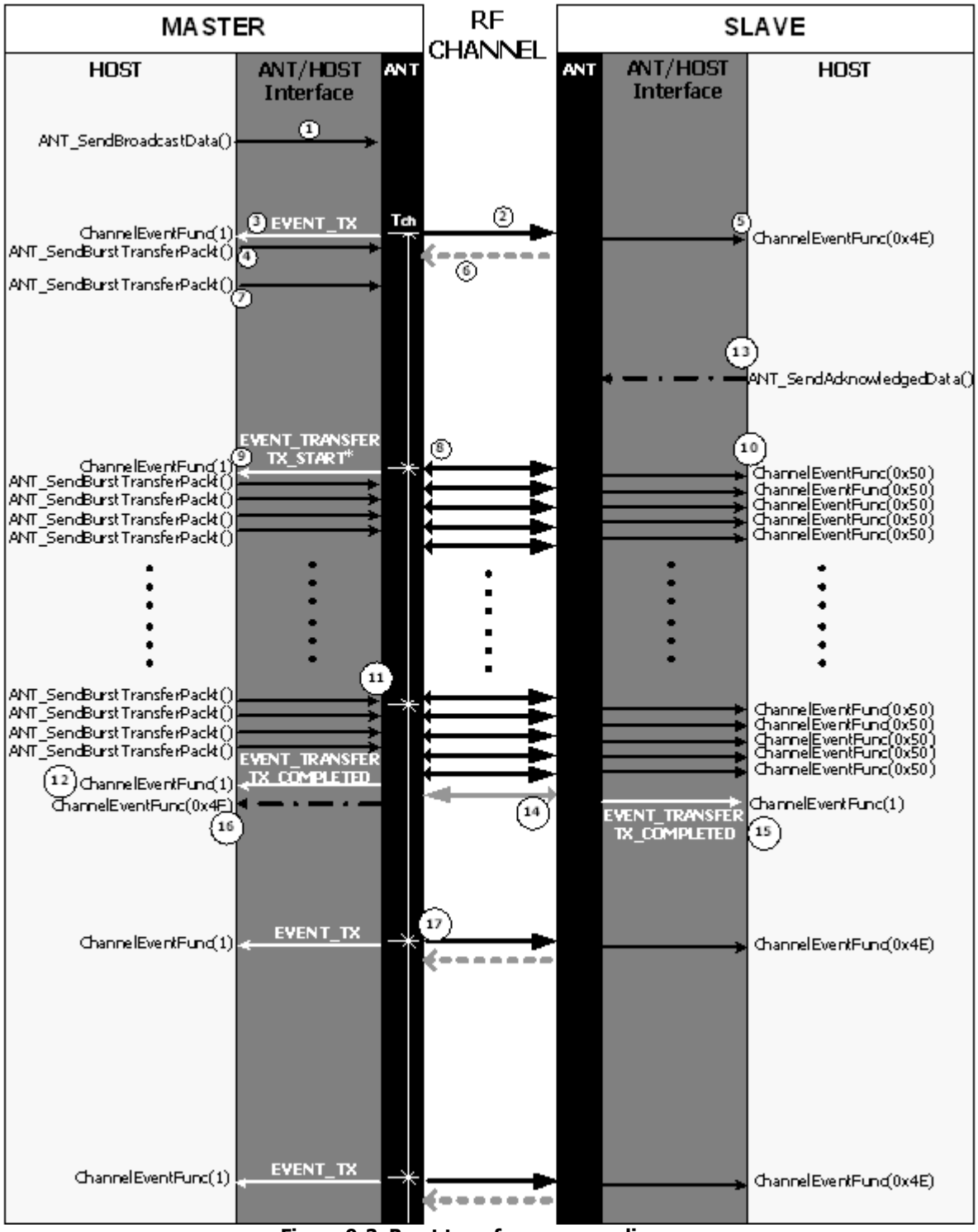


Figure 9-3. Burst transfer sequence diagram

In the example in Figure 9-3, assume the master's typical mode of operation is sending broadcast data to the slave. If the master wishes to send a large amount of data, the master's host can send multiple packets in fast succession, using the Burst Data message in place of a Broadcast or Acknowledged Data message.

Figure 9-3 (1) shows the master's host, in typical operation, sending a broadcast data message, which is transmitted at the beginning of the next channel period (2). The EVENT_TX (3) informs the host that ANT is ready for more data, and the host initiates the burst transfer request by sending an ANT_SendBurstTransferPacket() command (4). Meanwhile, the slave's host has been sent the ChannelEventFunc(0x4E) (5) and no data was sent back in the reverse direction (6).

Once a burst transfer starts transmitting (i.e. on the next channel period), data packets are transmitted at a very high rate. It is important that the Host/ANT interface can sustain the maximum 20kbps rate. In order to facilitate this transfer, it is possible to 'prime' the ANT buffers with 2 (or 8, depending on ANT device) burst packets prior to the next channel period. Figure 9-3 shows the host priming the ANT buffers with two ANT_SendBurstTransferPacket() messages (4&7). Please refer to the "Burst Transfers" application note for more information on burst queuing.

Once the transfer starts on the next channel period (8), an EVENT_TRANSFER_TX_START (9) will be issued (note this is only applicable for some ANT devices, refer to section 9.4.2), indicating that ANT has started sending packets and is ready for more data. The slave's host is informed with a ChannelEventFunc(0x50) (10).

The host's MCU is also notified for new data through hardware flow control. In asynchronous communication mode, the RTS line is toggled, whereas the SEN line is toggled in the synchronous communication mode. See the "Interfacing with ANT General Purpose Chipsets and Modules" document for more information on these lines.

Note that for each packet ANT sends over the RF channel, ANT receives an acknowledgement (indicated by the small arrow heads on (8) and subsequent arrows); however, this acknowledgement is not passed onto the host. ANT will automatically retry any failed packet transfer up to 5 times.

When a single packet burst is sent, it behaves identically to an acknowledged message, there are no retries associated with a single packet burst.

Burst transfers are synchronized off each other and are independent of the channel period. If a burst is long enough, it will override the subsequent channel periods (11). Once the burst transfer has completed, the host is notified with an EVENT_TRANSFER_TX_COMPLETED (12). Similar to the acknowledged data type, the master host could use this response as a prompt to send more data to ANT for transmission on the next channel period. In this example, the host does not send more data for transmission.

If a transmit was requested by the slave's host prior to the commencement of the burst (13), then that message will be sent in the reverse direction, at the end of the burst transfer (14). In this case, the request is for an acknowledged message, and the slave will receive an EVENT_TRANSFER_TX_COMPLETED (or failed) (15). The master ANT will notify and send data to the host with the ChannelEventFunc(0x4F) message (16). As the master host did not send any new data after receiving the response function (12), ANT will default to broadcasting on the next channel period (17). It will retransmit the last burst packet (i.e. the data in its buffer).

Notes:

If any packet still fails after 5 retries, ANT will terminate the burst transfer and the host will be notified with an EVENT_TRANSFER_TX_FAILED. If the application wishes to retry, it must restart the Burst Transfer sequence.

If a burst transfer fails in the forward direction (i.e. EVENT_TRANSFER_TX_FAILED), no reverse direction data can be sent by the slave. Any data the slave has to transmit will wait for the next channel period.

It should be noted that although the example in the figure shows only a master to slave (i.e. forward direction) burst transaction, burst transfers are also supported in the reverse direction. A Slave can burst in the reverse direction after a master broadcast, acknowledge or burst data transfer.

Sequence Numbers:

The upper three bits of the channel number field are used as a sequence number to ensure transfer integrity.

The transmit MCU must ensure that the sequence numbers are generated correctly in order for the ANT burst state machine to function correctly.

The first packet of a burst transfer will have a sequence number of %000. The sequence number is then incremented with %001 for each successive packet in the transfer rolling over back to %001, when a value of %011 is reached. The most significant bit of the sequence bits %100 is used as a flag to indicate the last packet in a Burst Transfer.

Example:

Channel = 3

- Packet # Channel Number
- %000 00011 (0x03)
- %001 00011 (0x23)
- %010 00011 (0x43)
- %011 00011 (0x63)
- %001 00011 (0x23)
- %110 00011 (0xC3) [Last Packet]

It should be noted that although the example in the figure shows only a master to slave (i.e. forward direction) burst transaction, burst transfers are also supported in the reverse direction.

The node receiving the burst packet does not need to keep track of the sequence numbers of the burst packets, as those are automatically checked by ANT. It is not possible to receive packets in an incorrect order, or to skip a packet. When a packet is missed, the burst fails; it does not skip the packet. From the point where the burst begins until the node receives the last packet with the complete bit set in the sequence number or EVENT_TRANSFER_RX_FAILED, all burst packets are in the correct order.

9.5.5.4 Advanced Burst Data (0x72)

BOOL ANT_SendAdvancedTransfer(UCHAR ucChannel, UCHAR* pucData, ULONG ulSize, UCHAR ucStdPacketsPerSerialMesg);

BOOL ANT_SendAdvancedBurstDataPacket(UCHAR ucChannelSeq, UCHAR* pucData, UCHAR ucStdPacketsPerSerialMesg); // Transmit

OR

ChannelEventFunc(Channel, EVENT_RX_ADV_BURST_PACKET); // Receive

On embedded platforms, the burst message may be processed as any other message received from ANT by processing the MESG_ADV_BURST_DATA_ID (0x72).

Data can be provided for an advanced burst in two ways: sending standard 8-byte Burst Data (0x50) messages or sending variable length Advanced Burst Data (0x72) messages. The number of data bytes to include in the Advanced Burst Data (0x72) messages is limited by the capabilities of the device.

Data received in an advanced burst is passed to the host using variable length Advanced Burst Data (0x72) messages. In order to ensure appropriate message processing, check the message length field.

Note:

The initial received packet is always a normal burst message (0x50).

Parameters	Type	Range	Description
Sequence Number	UCHAR (Bits 7:5)	As specified	The upper 3 bits of this byte are used as a sequence number to ensure transfer integrity (see Section 9.5.5.3 for details on the sequence number).
Channel Number	UCHAR (Bits 4:0)	0..MAX_CHAN-1	The lower 5 bits represent the channel number that the burst transfer is taking place on.
Data 0	UCHAR	0..255	The first data byte
..			
Data N	UCHAR	0..255	The last data byte.

```
// Example Usage
// Transmitter
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_TRANSFER_TX_COMPLETED:
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          ANT_SendAdvancedTransfer(Channel_0, DATA, 32, 3); // 24 bytes per packet, 32 bytes total
          break;
        }
      }
      break;
    }
  }
}
/*************************************************************/
// Receiver
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
```

```
  {
    case EVENT_RX_BURST_PACKET:  // PC applications only; use Msg ID 0x50 in embedded
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          // process initial received 8-byte normal burst packet.  If extended messages are enabled, extended
information is only present in this first packet
          break;
        }
      }
      break;
    }
    case EVENT_RX_ADV_BURST_PACKET:  // PC applications only; use Msg ID 0x72 in embedded
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          // process received data which is in channel event buffer one packet at a time validating the
          // sequence
          break;
        }
      }
      break;
    }
  }
}
```

Since advanced burst transfers offer the same functionality of standard bursts with additional enhancements, the mechanism for sending and receiving advanced burst transfers is very similar to that of normal burst transfers. Please refer to Section 9.5.5.3 for more details on normal burst data.

Figure 9-4 below describes the message transactions between host and ANT in both the master and slave device, and over the RF channel, when sending and advanced burst transfer from master to slave.

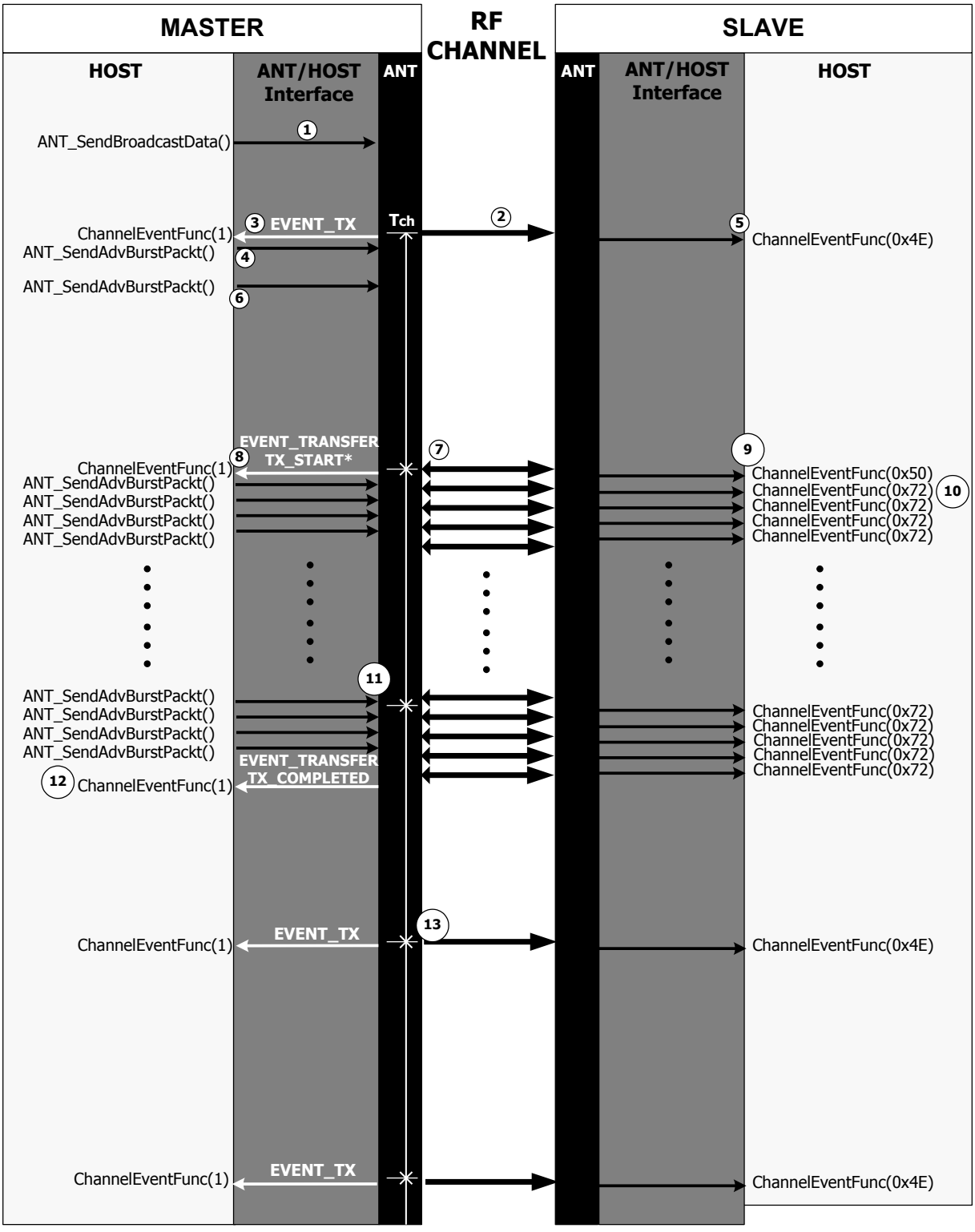


Figure 9-4. Advanced Burst Transfer Sequence Diagram

Similar to normal bursts, transmission begins at the start of the normal time slot and multiple data packets are sent consecutively. In the example in Figure 9-4, assume the master's typical mode of operation is sending broadcast data to the slave. If the master wishes to send a large amount of data, the master's host can send multiple packets in fast succession, using the Advanced Burst Data message in place of a Broadcast or Acknowledged Data message.

In Figure 9-4 (1) shows the master's host, in typical operation, sending a broadcast data message, which is transmitted at the beginning of the next channel period (2). The EVENT_TX (3) informs the host that ANT is ready for more data, and the host initiates the burst transfer request by sending an ANT_SendAdvancedBurstDataPacket() command (4). Meanwhile, the slave's host has been sent the ChannelEventFunc(0x4E) (5).

Once the transfer starts (7), an EVENT_TRANSFER_TX_START (8) will be issued, indicating that ANT has started sending packets and is ready for more data. The slave's host is informed with a ChannelEventFunc(0x50) for the first packet (9), and variable length ChannelEventFunc(0x72) for the rest of the packets in the burst (10). The size of the received advanced burst packets at the slave depends on the maximum packet size supported by both peers involved in an advanced burst transfer, and the size of the data transferred. For example, if a packet size of 24-bytes is being used, and the master sends a burst with 256 bytes of data, the slave will receive an initial 8-byte normal burst packet, followed by 10 24-byte advanced burst packets, and a final 8-byte advanced burst packet.

As with normal burst transfers, for each packet ANT sends over the RF channel, ANT receives an acknowledgement (indicated by the small arrow heads on (7) and subsequent arrows); however, this acknowledgement is not passed onto the host. ANT will automatically retry any failed packet transfer. The maximum number of retries depends on the configured retry count extension.

Advanced burst packets are synchronized relative to each other and are independent of the channel period. If a burst is long enough, it will override the subsequent channel periods (11). Similarly to normal burst transfers, once the transfer has completed, the host is notified with an EVENT_TRANSFER_TX_COMPLETED (12). Similar to the acknowledged data type, the master host could use this response as a prompt to send more data to ANT for transmission on the next channel period. In this example, the host does not send more data for transmission, so ANT will default to broadcasting on the next channel period (13), and will retransmit the data in its buffer. Unlike normal burst transfers, the retransmitted data will be the first packet of the burst (the initial 8-byte normal burst packet).

9.5.6 Channel Response / Event Messages

The Response/Event Messages are messages sent from the ANT device to the controller device, either in response to a message (see section 9.3 for a list of messages that generate responses), or as generated by an RF event on the ANT device.

9.5.6.1 Channel Response / Event (0x40)

ChannelEventFunc (Channel, MessageCode) // MessageID == 1

OR

ResponseFunc (Channel, MessageID) // MessageID != 1

The response/event message is either generated in response to a message or from an RF event.

Parameters	Type	Range	Description
Channel Number	UCHAR	0.. MAX_CHAN-1	The channel number of the channel associated with the event.
Message ID	UCHAR	0..255	ID of the message being responded too. This is set to 1 for an RF Event. (Message codes prefixed by EVENT_)
Message Code	enum	0..255	The response code or event code for a specific response or event

Message Codes* (The following message codes are defined in antdefines.h)

- Not all message Events are generated by all products. See section 9.4.2 for information on which event messages are supported by which products.
- Some channel events also include extended event parameters as detailed in section 0

Name	Value	Description
RESPONSE_NO_ERROR	0 (0x00)	Returned on a successful operation
EVENT_RX_SEARCH_TIMEOUT	1 (0x01)	A receive channel has timed out on searching. The search is terminated, and the channel has been automatically closed. In order to restart the search the Open Channel message must be sent again.
EVENT_RX_FAIL	2 (0x02)	A receive channel missed a message which it was expecting. This happens when a slave is tracking a master and is expecting a message at the set message rate.
EVENT_TX	3 (0x03)	A Broadcast message has been transmitted successfully. This event should be used to send the next message for transmission to the ANT device if the node is setup as a master.
EVENT_TRANSFER_RX_FAILED	4 (0x04)	A receive transfer has failed. This occurs when a Burst Transfer Message was incorrectly received.
EVENT_TRANSFER_TX_COMPLETED	5 (0x05)	An Acknowledged Data message or a Burst Transfer sequence has been completed successfully. When transmitting Acknowledged Data or Burst Transfer, there is no EVENT_TX message.

Name	Value	Description
EVENT_TRANSFER_TX_FAILED	6 (0x06)	An Acknowledged Data message, or a Burst Transfer Message has been initiated and the transmission failed to complete successfully
EVENT_CHANNEL_CLOSED	7 (0x07)	The channel has been successfully closed. When the Host sends a message to close a channel, it first receives a RESPONSE_NO_ERROR to indicate that the message was successfully received by ANT; however, EVENT_CHANNEL_CLOSED is the actual indication of the closure of the channel. As such, the Host must use this event message rather than the RESPONSE_NO_ERROR message to let a channel state machine continue.
EVENT_RX_FAIL_GO_TO_SEARCH	8 (0x08)	The channel has dropped to search mode after missing too many messages.
EVENT_CHANNEL_COLLISION	9 (0x09)	Two channels have drifted into each other and overlapped in time on the device causing one channel to be blocked.
EVENT_TRANSFER_TX_START	10 (0x0A)	Sent after a burst transfer begins, effectively on the next channel period after the burst transfer message has been sent to the device.
EVENT_TRANSFER_NEXT_DATA_BLOCK	17 (0x11)	Returned to indicate a data block release on the burst buffer.
CHANNEL_IN_WRONG_STATE	21 (0x15)	Returned on attempt to perform an action on a channel that is not valid for the channel's state
CHANNEL_NOT_OPENED	22 (0x16)	Attempted to transmit data on an unopened channel
CHANNEL_ID_NOT_SET	24 (0x18)	Returned on attempt to open a channel before setting a valid ID
CLOSE_ALL_CHANNELS	25 (0x19)	Returned when an OpenRxScanMode() command is sent while other channels are open.
TRANSFER_IN_PROGRESS	31 (0x1F)	Returned on an attempt to communicate on a channel with a transmit transfer in progress.
TRANSFER_SEQUENCE_NUMBER_ERROR	32 (0x20)	Returned when sequence number is out of order on a Burst Transfer
TRANSFER_IN_ERROR	33 (0x21)	Returned when a burst message passes the sequence number check but will not be transmitted due to other reasons.
MESSAGE_SIZE_EXCEEDS_LIMIT	39 (0x27)	Returned if a data message is provided that is too large.
INVALID_MESSAGE	40 (0x28)	Returned when message has invalid parameters
INVALID_NETWORK_NUMBER	41 (0x29)	Returned when an invalid network number is provided. As mentioned earlier, valid network numbers are between 0 and MAX_NET-1.
INVALID_LIST_ID	48 (0x30)	Returned when the provided list ID or size exceeds the limit.
INVALID_SCAN_TX_CHANNEL	49 (0x31)	Returned when attempting to transmit on ANT channel 0 in scan mode.
INVALID_PARAMETER_PROVIDED	51 (0x33)	Returned when invalid configuration commands are requested

Name	Value	Description
EVENT_SERIAL_QUE_OVERFLOW	52 (0x34)	This event indicates that the outgoing serial buffer of the USB chip has overflowed. This typically happens if the ANT chip is actively generating serial messages but the PC application is stalled/not running. This event is sent to notify the host application that the message buffer was full and some messages were lost
EVENT_QUE_OVERFLOW	53 (0x35)	May be possible when using synchronous serial port, or using all channels on a slow asynchronous connection. Indicates that one or more events were lost due to excessive latency in reading events out over the serial port. This typically happens if the serial queue is full.
NVM_FULL_ERROR	64 (0x40)	Returned when the NVM for SensRcore mode is full.
NVM_WRITE_ERROR	65 (0x41)	Returned when writing to the NVM for SensRcore mode fails.
USB_STRING_WRITE_FAIL	112 (0x70)	Returned when configuration of a USB descriptor string fails.
MESG_SERIAL_ERROR_ID	174 (0xAE)	This message is generated if the ANT chip receives a USB data packet that is not correctly formatted. The data portion of this message may be used to debug the USB packet. The first byte of the data (usually channel number) is the error number. 0 – the first byte of the USB data packet was not the ANT serial message Tx sync byte (0xA4) 2 – the checksum of the ANT message was incorrect 3 – the size of the ANT message was too large The rest of the data is a copy of the message that was sent.
ENCRYPT_NEGOTIATION_SUCCESS	0x38	When an ANT slave has negotiated successfully with an encrypted ANT master this event is passed to both the master and the slave.
ENCRYPT_NEGOTIATION_FAIL	0x39	When an ANT slave fails negotiation with an encrypted ANT master this event is passed to both the master and the slave. This can occur due to configuration mismatch, poor RF, mismatched encryption keys, or white/blacklisting by the master.

Name	Value	Description

```
// Example Usage
BOOL ANT_ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_RX_BROADCAST:
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          // process data which is in aucChannelEventBuffer
         break;
        }
        case Channel_N:
        {
          // process data which is in aucChannelEventBuffer
          break;
        }
      }
      break;
    }
    case EVENT_RX_FAIL:
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          // data packet was lost
          break;
        }
        case Channel_N:
        {
          // data packet was lost
          break;
        }
      }
      break;
    }
    case Default:
    {
      // catch unexpected message codes
      break;
    }
  }
}
```

9.5.6.2 Extended Event Parameters

- Certain events generated by ANT may include additional bytes after the message code
- The host can check the length of the event to determine if these additional fields were sent by ANT

9.5.6.2.1 ENCRYPT_NEGOTIATION_SUCCESS (0x38)

Parameters	Type	Description
Encryption ID	UCHAR[4]	The unique 4 byte identifier of the encrypted master or the negotiating slave.
User Info String (Optional)	UCHAR[19] (Optional)	The user information string, ANT only sends this to the host if the negotiating slave has enabled this capability. This parameter is not passed to the slave.

9.5.6.2.2 ENCRYPT_NEGOTIATION_FAIL (0x39)

Parameters	Type	Description
Encryption ID	UCHAR[4]	The unique 4 byte identifier of the encrypted master or the negotiating slave. In the case where negotiation fails due poor RF this parameter may not be passed.

9.5.7 Requested Response Messages

The following messages are returned in response to a Request Message (refer to section 9.5.4.4) sent to ANT. The specific response message sent is dependent on request's message ID parameter. The ANT PC library will call the Host application's ANT response function with the message ID as indicated below for each message.

The message ID codes are defined in antmessage.h.

9.5.7.1 Channel Status (0x52)

ResponseFunc (Channel, 0x52)

Parameters	Type	Range	Description
Channel Number	UCHAR	0.. MAX_CHAN-1	The channel number
Channel Status	UCHAR	Bits 4:7	Channel type (invalid on AP1 devices) Refer to Table 5-1
		Bits 2:3	Network number (invalid on AP1 devices) Refer to section 5.2.5.1.
		Bits 0:1	Channel State: Un-Assigned = 0 Assigned = 1 Searching = 2 Tracking = 3

```
// Example Usage
BOOL ANT_ResponseFunction(UCHAR ucChannel, UCHAR ucResponseMesgID)
{
  Switch (ucResponseMesgID)
  {
    case MESG_CHANNEL_STATUS_ID:
    {
      switch (aucResponseBuffer[1] & 0x3) // channel status
      {
        case 0:
        {
          // channel is un-assigned
          break;
        }
        case 1:
        {
          // channel is assigned
          break;
        }
      }
      break;
    }
  }
}
```

9.5.7.2 Channel ID (0x51)

ResponseFunc (Channel, 0x51)

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The channel number
Device Number	USHORT (little endian)	0..65535	The device number
Device Type ID	UCHAR	0..127	The device type
Transmission Type	UCHAR	0..255	The transmission type

This message returns the channel ID of the specified channel. This message is useful when trying to pair devices. When a slave is attempting to pair with a master, it will typically set one or more of the device number, device type, or transmission type fields with a wild card. When the slave finds a device that matches the search – by successfully receiving data, the Request Message can be used to return the discovered channel's ID. This ID can then be saved for future use in opening the channel and searching for this specific device. Refer to section 6 for more information.

Note that the transmission type and device type IDs are assigned and regulated to maintain network integrity, and interoperability, except for the default public network. Please visit www.thisisant.com for more details on available standard network types or on how to obtain your own network type identifier.

9.5.7.3 ANT Version (0x3E)

ResponseFunc (-, 0x3E)

The version message returns an N-byte null-terminated version string, corresponding to the ANT host interface version.

Parameters	Type	Range	Description
Version Message	char[11]	1..255	Variable length string.

Please note that this message is not supported on all ANT products. Refer to section 9.4 for capabilities. The length of the ANT version message is part specific, and can be checked by requesting this string.

9.5.7.4 Capabilities (0x54)

ResponseFunc (-, 0x54)

This message returns a summary of the ANT device's configuration, which is dependent on both the software embedded in the ANT MCU and on hardware limitations.

Parameters	Type	Range	Description
Max ANT Channels	UCHAR	0..MAX_CHAN	Returns the Number of ANT channels available
Max Networks	UCHAR	0..MAX_NET-1	Returns the number of networks available
Standard Options	UCHAR	0..255	The Standard Options bitfield is encoded as follows: Bit 0 - CAPABILITIES_NO_RECEIVE_CHANNELS Bit 1 - CAPABILITIES_NO_TRANSMIT_CHANNELS Bit 2 - CAPABILITIES_NO_RECEIVE_MESSAGES Bit 3 - CAPABILITIES_NO_TRANSMIT_MESSAGES Bit 4 - CAPABILITIES_NO_ACKD_MESSAGES Bit 5 - CAPABILITIES_NO_BURST_MESSAGES Other bits are reserved
Advanced Options	UCHAR	0..255	The Advanced Options bit field is encoded as follows: Bit 1 - CAPABILITIES_NETWORK_ENABLED Bit 3 - CAPABILITIES_SERIAL_NUMBER_ENABLED Bit 4 - CAPABILITIES_PER_CHANNEL_TX_POWER_ENABLED Bit 5 - CAPABILITIES_LOW_PRIORITY_SEARCH_ENABLED Bit 6 - CAPABILITIES_SCRIPT_ENABLED Bit 7 - CAPABILITIES_SEARCH_LIST_ENABLED Other bits are reserved
Advanced Options 2	UCHAR	0..255	The Advanced Options 2 bitfield is encoded as follows: Bit 0 - CAPABILITIES_LED_ENABLED Bit 1 - CAPABILITIES_EXT_MESSAGE_ENABLED Bit 2 - CAPABILITIES_SCAN_MODE_ENABLED Bit 4 - CAPABILITIES_PROX_SEARCH_ENABLED Bit 5 - CAPABILITIES_EXT_ASSIGN_ENABLED Bit 6 – CAPABILITIES_FS_ANTFS_ENABLED Other bits are reserved
Advanced Options 3	UCHAR	0..255	The Advanced Options 3 bitfield is encoded as follows: Bit 0 – CAPABILITIES_ADVANCED_BURST_ENABLED Bit 1 – CAPABILTIES_EVENT_BUFFERING_ENABLED Bit 2 – CAPABILITIES_EVENT_FILTERING_ENABLED Bit 3 – CAPABILITIES_HIGH_DUTY_SEARCH_ENABLED Bit 6 – CAPABILITIES_SELECTIVE_DATA_UPDATES_ENABLED Other bits are reserved

9.5.7.5 Device Serial Number (0x61)

ResponseFunc (-, 0x61)

Parameters	Type	Range	Description
Serial Number	char[4]	1..255	4 byte serial number

Please note this message is only available on specific devices, refer to section 9.4for capabilities. The serial number is a 4-byte, little-endian encoded unsigned integer.

9.5.7.6 Event Buffer Configuration (0x74)

Response Func(-, 0x74)

Parameters	Type	Range	Description
Filler	UCHAR	0	A filler 0 byte.
Buffer Config	UCHAR	0..255	0x00 - Buffer Low Priority Events† 0x01 - Buffer all Events
Buffer Size	USHORT	‡	Bytes received before a buffer flush occurs
Buffer Time	USHORT	0..0xFFFF	Time in 10ms units before a buffer flush occurs

†EVENT_TX, EVENT_RX_FAIL, and EVENT_CHANNEL COLLISION only
‡The Buffer Size may be a value between 0 and a device specific maximum

Please note that this message is not supported on all ANT products, refer to section 9.4. The Event Buffer message returns the current event buffering settings. These settings may be configured using the Configure Event Buffer command (9.5.2.22).

9.5.7.7 Advanced Burst Capabilities (0x78)

ResponseFunc (-, 0x78)

This message returns a summary of the ANT device's advanced burst capabilities.

Parameters	Type	Range	Description
Message Type	UCHAR	0	If this field is set to 0, this message contains the capabilities of the device for advanced burst.
Supported Max Packet Length	UCHAR	As specified	Specifies the maximum burst packet size that can be sent or received by the device: 0x01 – 8-byte 0x02 – 16-byte 0x03 – 24-byte
Supported Features	ULONG (3 bytes, little endian	As specified	The Supported Options bitfield is encoded as follows: Bit 0 – ADV_BURST_FREQUENCY_HOP_ENABLED Other bits are reserved – set to zero.

```
// Example Usage
ANT_RequestMessage(0, MESG_ADVANCED_BURST_CAPABILITIES_CONFIG); // request the advanced burst capabilities of the device
// response message will contain the advanced burst capabilities; no RESPONSE_NO_ERROR will be sent by ANT
```

The frequency hopping feature provides increased immunity to RF interference by rotating through 6 RF frequencies on each transaction pair (data packet and acknowledged packet) during the transfer. The set of frequencies used is

automatically selected by ANT; no additional configuration is required on the application level other than enabling this feature.

9.5.7.8 Advanced Burst Current Configuration (0x78)

ResponseFunc(-, 0x78)

This message returns the current advanced burst configuration, as set by the Configure Advanced Burst Command (0x78).

Parameters	Type	Range	Description
Message Type	UCHAR	1	If this field is set to 1, this message contains the current advanced burst configuration.
Enable	UCHAR	0..1	Specifies whether advanced burst is enabled or disabled: 0x00 – Disabled 0x01 - Enabled
Max Packet Length	UCHAR	As specified	Specifies the maximum burst packet size that will be sent or received by the device: 0x01: 8 byte 0x02: 16 byte 0x03: 24 byte
Required Features	ULONG (3 bytes, little endian)	As specified	Required advanced burst features. See "Supported Features" field in section 9.5.7.7 for a list of available features.
Optional Features	ULONG (3 bytes, little endian)	As specified	Optional advanced burst features. See "Supported Features" field in section 9.5.7.7 for a list of available features.
Stall Count [optional]	USHORT (little endian)	0..65535	Number of stall packets that will be sent before a transfer enters the retry count. Each packet corresponds to ~3ms. Typical default value is 3210 to provide ~10s of stalling.
Retry Count Extension [optional]	UCHAR	0..255	Number of retry count (5 retries) cycle extensions. Typical default value is 4, providing 20 retries.

// Example Usage
ANT_RequestMessage(1, MESG_ADVANCED_BURST_CAPABILITIES_CONFIG); // request the advanced burst current configuration of the device
// response message will contain the advanced burst capabilities; no RESPONSE_NO_ERROR will be sent by ANT			

9.5.7.9 Event Filter (0x79)

Response Func (-, 0x79)

Parameters	Type	Range	Description
Channel Number	UCHAR	0	Set to 0x00
Event Filter	USHORT (2 bytes, little endian)	0..65535	The Event Filter bit fields is as follows: Bit 0 – Filter event 1 (RESPONSE_NO_ERROR) Bit 1 – Filter event 2 (EVENT_RX_SEARCH_TIMEOUT) … Bit N – Filter event (N+1) where N is max 15 Setting a bit to 1 applies a filter to the corresponding event.

// Example Usage
ANT_RequestMessage(0, MESG_CONFIG_EVENT_FILTER); // request the event filter applied to the specified channel
// response message will contain the channel number and event filter; no RESPONSE_NO_ERROR will be sent by ANT			

Please note that this message is only available on specific devices as listed in section 9.4. The Event Filter message returns the current event filter settings. These settings may be configured using the Configure Event Filter command (9.5.2.26).

9.5.7.10 Selective Data Update (SDU) Mask Setting (0x7B)

Response Func (-, 0x7B)

Parameters	Type	Range	Description
Sub Message ID: SDU Mask Number	UCHAR	0..MAX_SDU_MASKS	The SDU mask number associated with the SDU Mask
SDU Mask (8 bytes)	UCHAR[8]	N/A	Defines which bits in an 8 byte message are compared for selective data update purposes. 0 – Ignore 1 – Compare and send data update when this data changes

// Example Usage
ANT_RequestMessage(SDU Mask Number, MESG_SET_SDU_MASK); // request the SDU mask associated with the specified SDU mask number
// response message will contain the SDU mask number and the SDU mask; no RESPONSE_NO_ERROR will be sent by ANT			

Please note that this message is only available on specific devices as listed in section 9.4. The SDU Mask Setting message returns the current SDU Mask. This is configured using the Set SDU Mask command (0).

9.5.7.11 User NVM (0x7C)

Response Func (-, 0x7C)

Parameters	Type	Range	Description
Filler	UCHAR	0	Set to zero.
Data	UCHAR[]	Variable	Variable length user data.

Please note that this message is not supported on all ANT products, refer to section 9.4. The User NVM message returns the requested data from User NVM. The starting address and size of the data correspond to the values specified in the Request Message. Read block size may be arbitrary up to the device specific maximum. It is not possible to read beyond any address in the user space.

9.5.7.12 Single Channel Encryption Parameters (0x7D)

ResponseFunc (Parameter, 0x7D)

Parameters	Type	Range	Description
Requested Encryption Parameter	UCHAR	As specified	0x00 – Max Supported Encryption Mode 0x01 – Encryption ID 0x02 – User Information String
Max Supported Encryption Mode or Encryption ID or User Info String	UCHAR or UCHAR[4] or UCHAR[19]	0..2 or N/A or N/A	The maximum encryption mode supported or The unique 4 byte identifier used for encryption mode or The user information string
// Example Usage ANT_RequestMessage(1, MESG_CRYPTO_PARAMETER) //Request the 4 byte encryption ID			

The max supported encryption mode parameter indicates all lower value modes are also supported, for example a max supported encryption mode value of 2 implies that encryption mode 1 is also supported. (Refer to section 9.5.2.30 for information on encryption modes.)

9.5.8 Test Mode

9.5.8.1 Init CW Test Mode (0x53)

BOOL ANT_InitCWTestMode(void);

Parameters	Type	Range	Description
Filler	UCHAR	0	

This function must be called before the CW Test Mode message below in order to initialize the module to the correct state for CW mode.

Note: This command should be executed only directly after a reset, or a System Reset command. Failure to do so may result in unpredictable results.

9.5.8.2 CW Test Mode (0x48)

BOOL ANT_SetCWTestMode(UCHAR ucTransmitPower, UCHAR ucRFChannel);

Parameters	Type	Range	Description
Filler	UCHAR	0	A filler 0 byte that must be included
Transmit Power	UCHAR	0..4	Refer to section 9.4.3.
Channel RF Frequency	UCHAR	0..127	Channel Frequency = 2400 MHz + Channel RF Frequency Number * 1.0 MHz

// Example Usage
ANT_InitCWTestMode();
// wait for RESPONSE_NO_ERROR
ANT_SetCWTestMode(3, 57); // set RF power to 0dBm and CW 2457MHz			

This message is used to put the radio into a CW test mode using a given transmit power level and channel RF frequency.

This command is intended to test your implementation for RF regulatory requirements. It will set ANT to transmit an unmodulated carrier wave on the specified RF frequency, at the specified power level.

Note: This command should be executed only directly after an Init CW Test Mode (0x53) command as described above. Failure to do so may result in unpredictable results.

9.5.9 Extended Data Messages

Each of the Data Message functions described in section 0 can be sent in the legacy extended data message format. These functions are now supported in nRF24AP2 and CC257x modules as flagged extended message bytes in existing data messages. Refer to section 7.1.1. However, AP2 and CC257x ANT can still accept the data messages as described here.

9.5.9.1 Extended Broadcast Data (0x5D)

BOOL ANT_SendExtBroadcastData(UCHAR ucChannel, UCHAR* pucBroadcastData); // Transmit

or

ChannelEventFunc (Channel, EVENT_RX_EXT_BROADCAST) // Receive

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The channel the data is received from
Device Number	USHORT	0..65536	The device number
Device Type	UCHAR	0..255	The device type
Transmission Type	UCHAR	0..255	The transmission type
Data 0	UCHAR	0..255	The first data byte
Data N	UCHAR	0..255	The Nth data byte
Data 7	UCHAR	0..255	The eighth data byte

```
// Example Usage

// Transmitter
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_TX:
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          ANT_SendExtBroadcastData(Channel_0, DATA);
          break;
        }
      }
      break;
    }
  }
}

/*********************************************************/
// Receiver
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_RX_EXT_BROADCAST:  // PC applications only; use MsgID 0x5D in embedded
    {
      switch (ucChannel)
```

```
      {
        case Channel_0:
        {
          // process received data which is in channel event buffer
          break;
        }
      }
      break;
    }
  }
}
```

The legacy extended broadcast functions the same way as normal broadcast, except that the Channel ID is appended to the front of the data. Extended messages are enabled by default when Rx Scan Mode is being used.

Receiver

The corresponding channel slave receives the data at its programmed channel period and generates a legacy Extended Broadcast Data message to its MCU. If the slave does not manage to receive a data packet for its time slot, an EVENT_RX_FAIL will be generated instead.

If you are using the ANT library interface it will fill the data into your receive buffer, then send a special library-only event EVENT_RX_EXT_BROADCAST to let you know that a valid extended broadcast message has been received.

9.5.9.2 Extended Acknowledged Data (0x5E)

BOOL ANT_SendExtAcknowledgedData(UCHAR ucChannel, UCHAR* pucBroadcastData); // Transmit

or

ChannelEventFunc(Channel, EVENT_RX_EXT_ACKNOWLEDGED) // Receive

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The channel the data is for/from
Device Num	USHORT	0..65536	The device number
Device Type	UCHAR	0..255	The device type
Transmission Type	UCHAR	0..255	The transmission type
Data 0	UCHAR	0..255	The first data byte
Data N	UCHAR	0..255	The Nth data byte
Data 7	UCHAR	0..255	The eighth data byte

```
// Example Usage
// Transmitter
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_TRANSFER_TX_COMPLETED:
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          ANT_SendExtAckknowledgedData(Channel_0, DATA);
          break;
        }
      }
      break;
    }
  }
}

/*******************************************************/
// Receiver
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_RX_EXT_ACKNOWLEDGED:  // PC applications only; use MsgID 0x5E in embedded
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          // process received data which is in channel event buffer
          break;
        }
      }
      break;
    }
  }
}
```

Extended acknowledged data functions the same way as normal acknowledge, except that the Channel ID is appended to the front of the data. Extended messages are enabled by default when Rx Scan Mode is being used.

Receiver

Reception of Acknowledged Data from the master causes an Extended Acknowledged Data message to be sent to the slave MCU. If the message reception fails, an EVENT_RX_FAIL occurs.

If you are using the ANT library interface it will fill the data into your receive buffer, then send a special library only event EVENT_RX_EXT_ACKNOWLEDGED to let you know that a valid extended acknowledge message has been received.

9.5.9.3 Extended Burst Data (0x5F)

BOOL ANT_SendExtBurstTransfer(UCHAR ucChannel, UCHAR* pucData, USHORT usNumDataPackets); // Transmit

BOOL ANT_SendExtBurstTransferPacket(UCHAR ucChannelSeq, UCHAR* pucData); // Transmit

or

ChannelEventFunc (Channel, EVENT_RX_EXT_BURST_PACKET) // Receive

Parameters	Type	Range	Description
Sequence Number	UCHAR (Bits 7:5)	As specified	The upper 3 bits of this byte are used as a sequence number to ensure transfer integrity (see below).
Channel Number	UCHAR (Bits 4:0)	0..MAX_CHAN-1	The lower 5 bits are the channel number the burst transfer is taking place on.
Device Num	USHORT	0..65536	The device number
Device Type	UCHAR	0..255	The device type
Transmission Type	UCHAR	0..255	The transmission type
Data 0	UCHAR	0..255	The first data byte
Data N	UCHAR	0..255	The Nth data byte
Data 7	UCHAR	0..255	The eighth data byte

```
// Example Usage
// Transmitter
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_TRANSFER_TX_COMPLETED:
    {
      switch (ucChannel)
      {
        case Channel_0:
        {
          ANT_SendExtBurstData(Channel_0, DATA, 4); // 8 bytes per packet, 32 bytes total
          break;
        }
      }
      break;
    }
  }
}

/*************************************************************/
// Receiver
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_RX_EXT_BURST_PACKET:  // PC applications only; use MsgID 0x5F in embedded
    {
      switch (ucChannel)
      {
```

Parameters	Type	Range	Description

```
        case Channel_0:
        {
          // process received data which is in channel event buffer one packet at a time validating the
          // sequence
          break;
        }
      }
      break;
    }
  }
}
```

Extended burst data functions the same way as normal burst data, except that the Channel ID is appended to the front of the data. Extended messages are enabled by default when Rx Scan Mode is being used.

Receiver

Reception of Burst Data from the master causes Extended Burst Data Messages to be sent to the slave MCU. If burst message reception exceeds the maximum number of retries an EVENT_TRANSFER_RX_FAIL occurs.

9.5.10 PC Functional Interface Configuration

The functions described in this section are unique to the ANT PC Library interface, and are used to set up and configure the ANT PC Library for use. They are not available to an embedded application as the messages are exchanged directly through a serial interface.

9.5.10.1 ANT PC Library Usage Notes

The following notes apply when using the ANT PC Library. The files for this library can be downloaded from www.thisisant.com:

- ANT_DLL.dll, DSI_CP210xManufacturing_3_1.dll and DSI_SiUSBXp_3_1.dll must be accessible to the application that is using the ANT PC Library. In other words, these files must be placed in the same folder as the executable, or in a Windows system folder.
- antmessage.h and antdefines.h must be included where calls to the ANT PC Library are made.

9.5.10.2 ANT_Init

BOOL ANT_Init(UCHAR ucUSBDeviceNum, USHORT usBaudrate);

Parameters	Type	Range	Description
ucUSBDeviceNum	UCHAR	0..N-1	USB device number of the module to connect to. Modules connected to a PC will be assigned USB device numbers starting from 0. N is the number of USB ANT devices that are connected.
usBaudrate	USHORT		Asynchronous baud rate used to connect to the ANT controller. See specific ANT controllers for allowable baud rates.

```
// Example Usage
if (ANT_Init(0, 38400) == false)
  // error message
else
  // continue to ANT initialization
```

ANT_Init is called to initialize the ANT library and connect to the ANT module. Function returns TRUE if successfully connected to the module, otherwise returns FALSE.

9.5.10.3 ANT_Close

void ANT_Close (void);

Parameters	Type	Range	Description
None			

```
// Example Usage
ANT_Close();
```

ANT_Close() closes the USB connection to the ANT module.

9.5.10.4 ANT_AssignResponseFunction

void ANT_AssignResponseFunction(RESPONSE_FUNC pfResponse, UCHAR *pucResponseBuffer);

Parameters	Type	Description
pfResponse	RESPONSE_FUNC	Pointer to the function that will be called whenever a response / event message is received from the module.
pucResponseBuffer	UCHAR*	Pointer to the buffer where the data of the response / event message will be written to. This buffer should be sized to MESG_RESPONSE_EVENT_SIZE.

```
// Example Usage
BOOL ANT_ResponseFunction(UCHAR ucChannel, UCHAR ucResponseMesgID);
UCHAR aucResponseBuffer[MESG_RESPONSE_EVENT_SIZE];
..
ANT_AssignResponseFunction(&ANT_ResponseFunction, aucResponseBuffer);
```

ANT_AssignResponseFunction sets the response callback function and the return data buffer. The callback function and data buffer are used whenever a response message is received from ANT. The response buffer needs to be large enough to hold an incoming response, which is of size MESG_RESPONSE_EVENT_SIZE. This function must be called immediately after calling ANT_Open and before any other ANT calls are made.

The response function must be a C function.

9.5.10.5 ANT_AssignChannelEventFunction

void ANT_AssignChannelEventFunction(UCHAR ucChannel, CHANNEL_EVENT_FUNC pfChannelEvent, UCHAR *pucRxBuffer);

Parameters	Type	Description
ucChannel	UCHAR	Channel Number
pfChannelEvent	CHANNEL_EVENT_FUNC	Pointer to the function that will be called whenever an event for this channel occurs.
pucResponseBuffer	UCHAR*	Pointer to the buffer where the data of the response/event message is written. This buffer should be sized to MESG_DATA_SIZE.

```
// Example Usage
BOOL ANT_ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent);
UCHAR aucChannelEventBuffer[MESG_DATA_SIZE];

ANT_AssignChannelEventFunction(channel_0, &ANT_ChannelEventFunction, aucChannelEventBuffer);
```

ANT_AssignChannelEventFunction sets the channel event function and the return data buffer. The callback function and data buffer are used whenever an event message is received from ANT for the given channel. The response buffer needs to be large enough to hold an incoming response which is of size MESG_DATA_SIZE. This function must be called to set up a given channel before any other ANT functions that use this channel are called.

The channel event callback function must be a C function. Each channel can have its own event callback function, along with a unique data buffer; or they can both be shared, or any combination thereof, that best suits the application.