ANT Message Protocol and Usage

Copyright Information and Usage Notice

Table of Contents

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, 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 reducing 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.
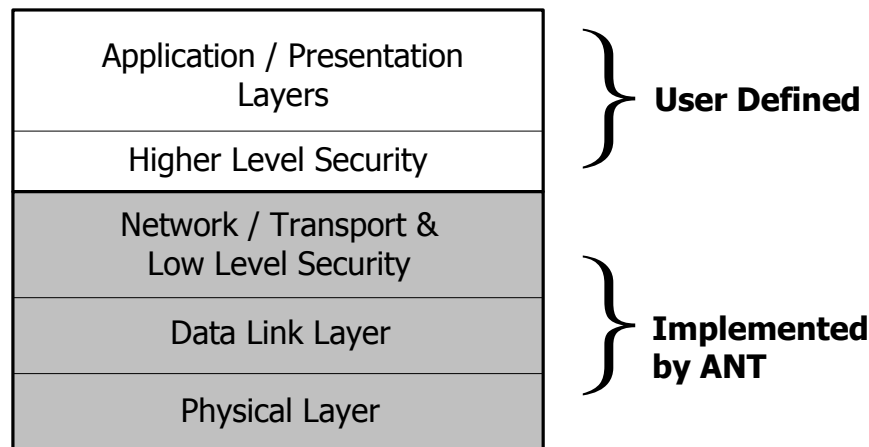
Figure 1-1. OSI Layer model of ANT

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.

ANT technology is available in the following formats:

ANT Single Chip & Chipset

Intended for integration onto the customer's PCB and interfaced with a host MCU.

1. Nordic Semiconductor nRF24AP2 chip family (-1ch, -8ch and –USB) – second generation of the ANT implementation integrated into a single-chip RF protocol and IC.

2. Nordic Semiconductor nRF24AP1 – first generation, complete ANT implementation integrated into a single-chip RF protocol and transceiver Integrated Circuit (IC).

3. AT3 chipset family – two-chip ANT solution that combines an ANT-protocol MCU with a Nordic Semiconductor RF IC (nRF24L01+ or nRF24L01).

ANT Module

The ANT modules are certified or certification ready PCB modules incorporating an ANT chip or chipset and can be mounted onto existing PCB, allowing for immediate product integration with minimal effort.

ANT USB Stick

The ANT USB Stick provides a bridge between an ANT network and a PC. ANT USB stick comes with royalty-free drivers which can be redistributed with ANT.

ANT Development Kit

Development Kits are available to provide a timely and efficient path to ANT integration for both the embedded and PC environments. The embedded environment offers easy integration with custom hardware. The PC environment provides USB interconnection along with drivers and sample applications.

ANT PC Interface Software

A royalty-free PC software library provides an interface to the ANT USB Stick and ANT Development Kit, and is readily integrated with a customer's PC application.

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.
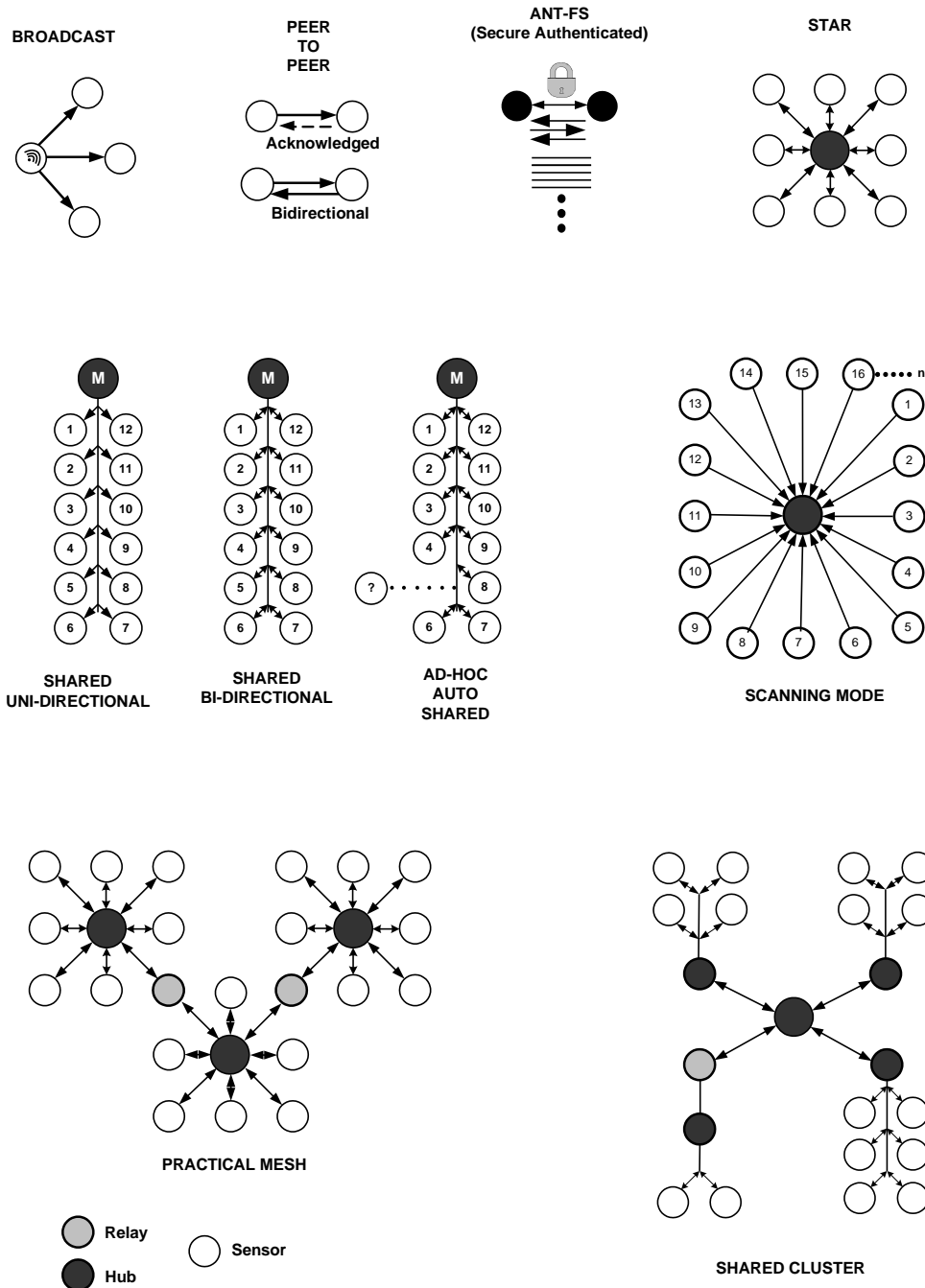
Figure 3-1. Example ANT Networks

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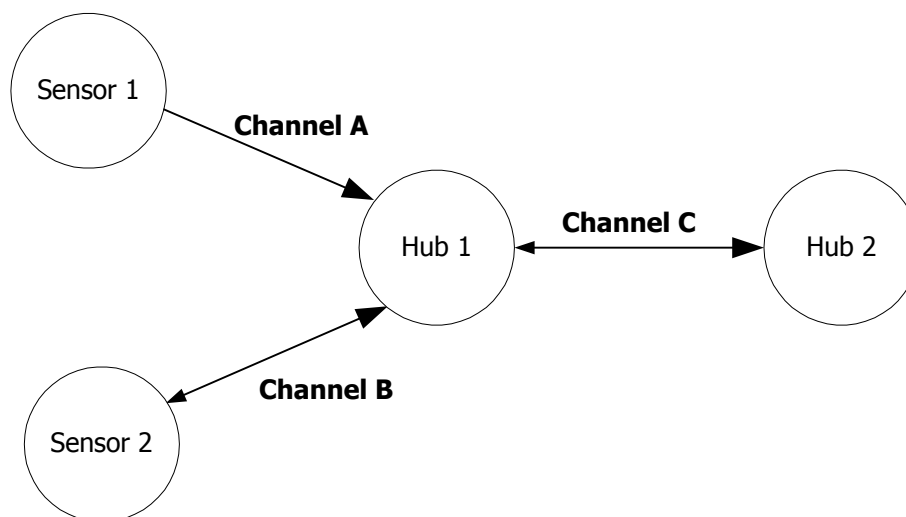
Figure 3-2. A Simple ANT Network

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 in fact 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. In Figure 3-2, large arrows indicate the primary data flow from master to slave, with small arrows indicating 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.

The following table describes the master / slave status of each of the channels shown in Figure 3-2

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 the following diagram.
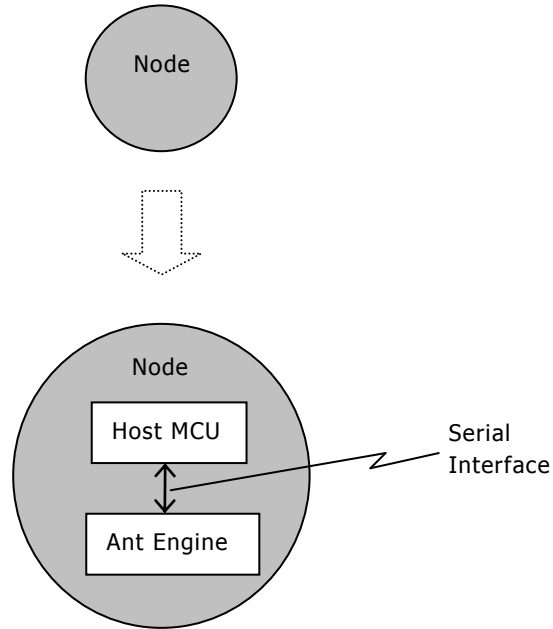
Figure 4-1. Contents of an ANT node

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.
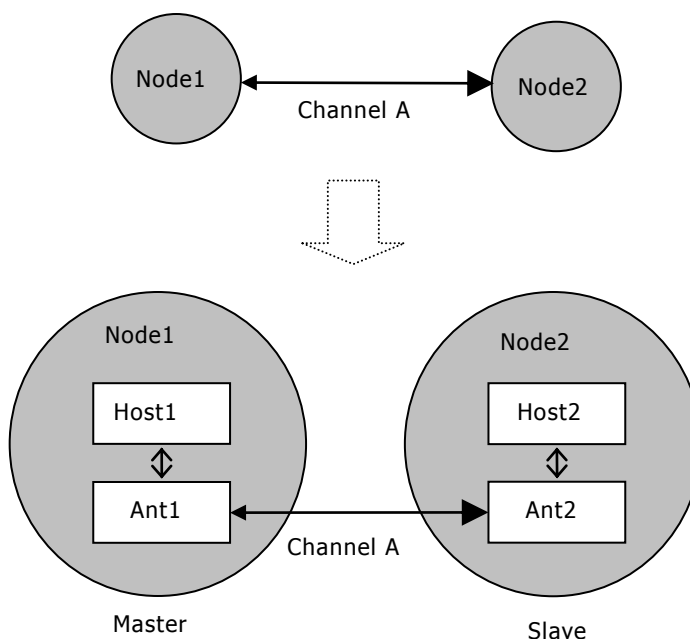
Figure 5-1. Channel communication between two ANT nodes

A channel consists of:

- A master (e.g. Node1)

- A slave (e.g. Node2)

5.1 Channel Communication

The ANT data types determine the type of communication that will occur between the two nodes of an ANT channel. There are three data types: broadcast, acknowledged and 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->ANT serial interface and messaging will be described in later sections.

The overall communication has two levels – one governs the direction (master to slave or vice versa) and the second specifies the type. They are described in detail in the following sections.

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

1. Forward Direction (Master -> Slave)

2. Reverse Direction (Slave -> Master)

Messages are transmitted in the forward direction at the designated channel period (Tch). 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. The slave may optionally send data back to the master in the reverse direction.
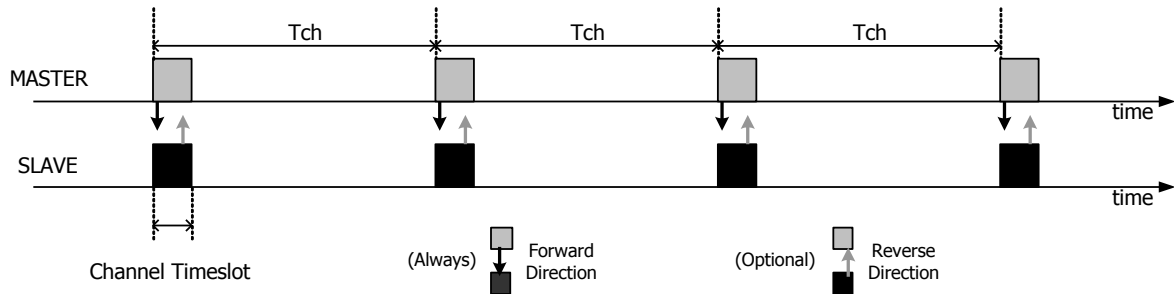
Figure 5-2. Channel communication showing forward and reverse directions. Not to scale.

There are three basic data types supported in both forward and reverse directions: broadcast, acknowledged and burst. These are described in section 5.3.

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

    - 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 below.

Value	Description
0x00	Bidirectional Slave Channel
0x10	Bidirectional Master Channel
0x20	Shared Bidirectional Slave Channel
0x40	Slave Receive Only Channel

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 mode 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 Sections 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 3-1. See section 5.5 and 5.6 for more information regarding independent and shared channels respectively.

5.2.1.3   Transmit/Receive Only Channel

Transmit and receive only channel types can only send data in the forward direction. In other words, the master cannot receive data from any slave and similarly, for the receive only channel, 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. Transmit only channels exist for legacy support and are not recommended for general use as it also disables the ANT channel management mechanisms. Receive only channels are recommended for diagnostic applications using continuous scanning 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 and background scanning channel. 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 Channel

The background scanning channel is a channel that is performing a continuous search operation. As with standard ANT search, it can be performed in either high or low priority modes. Only one background search channel 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 channel search timeouts are configured for low priority search mode. This will ensure that the background search mechanism does not interfere with any other channels operating on the device.

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

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, which 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. The following equation can be used to determine the value for the RF frequency field.

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

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 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. An example usage is the SensRcore™ implementation, which defines the two least significant bits of the transmission type 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) to indicate the presence of a Global Data Identification Byte.

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 transmission type can be defined as desired.

5.2.3.2   Device Type (+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 1, 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 section 1. The channel ID functional description is located in Section 9.5.2.3.

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 the following equation.

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

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 good, 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 effect the time between whole burst transfers, but does not effect 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 behaviors 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 (Please refer to Section 9 for more details). The default Network Number is 0. And network number 0 is assigned to the "Public Network" by default. For AP1 devices, the remaining network numbers are left uninitialised; however, for non-AP1 devices all network numbers also default to the public network key.

The network number will be assigned a network key using the Set Network Key (0x46) message, and any individual channels assigned to a network number will be using 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. Note, 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 below:

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 shows 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.
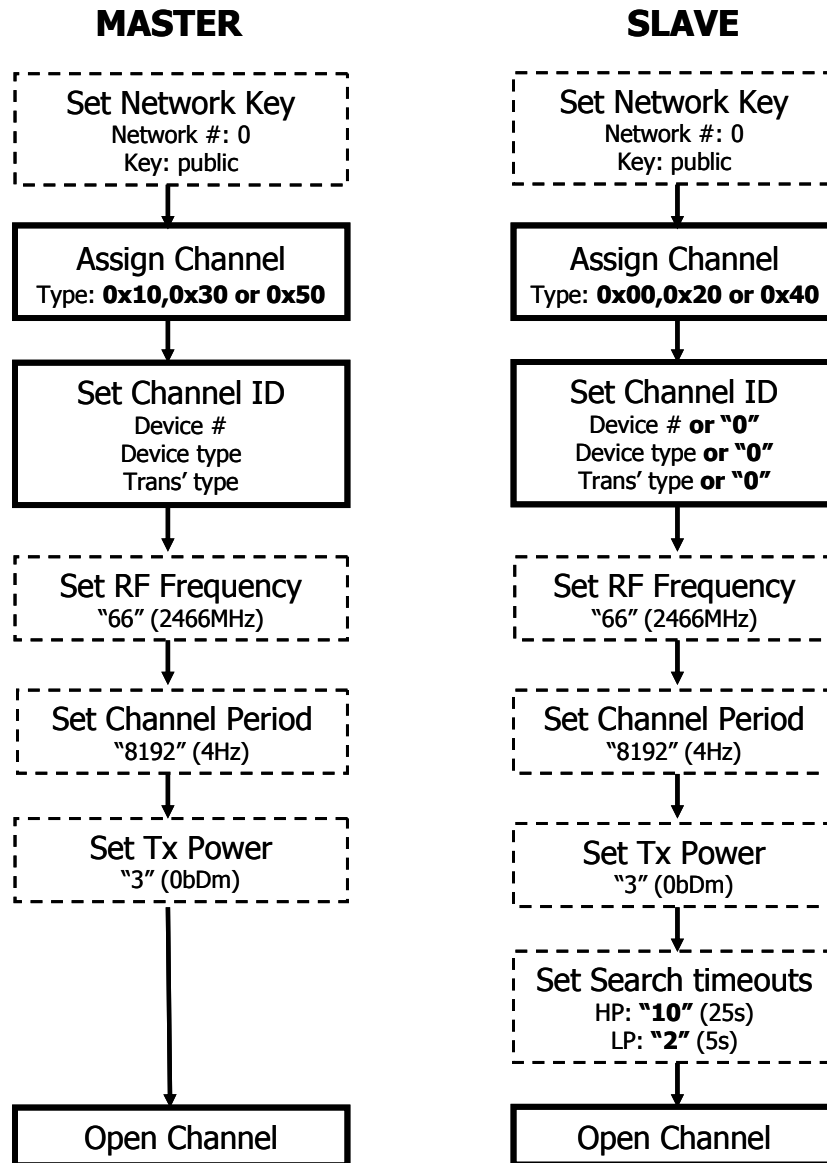
MASTER   **SLAVE**

Figure 5-3. Process to establish communication 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, this parameter must be set **prior to setting any other channel parameters.** Once the network key has been set, all other channel parameters will return to their default values. 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 as one of the transmit channel types, and

the slave node assigned a corresponding receive channel type. 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 (9.5.2.4), and the yet to be discussed Tx power (9.5.2.8) and search timeouts (9.5.2.5 & 9.5.2.12) 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, once its channel is opened, 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 three data types supported by ANT: broadcast, acknowledged and 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 three 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 uni-directional 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, and 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 bi-directional 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 off each other, not 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.

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. Some examples of this are:

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. Hence, 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 application note "Burst Transfers".

All data types are explained in further detail, and sequence diagrams provided, in section 9.5.5. 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.

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. 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 Not re-transmitted
Burst	Forward	A burst transfer will commence at start of the next timeslot. Bursts packets synchronize off each other
	Reverse	Same as above

All data types can also be 'extended' such that the receiving node's ANT will pass the channel ID 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. But 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, as 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, also, that the data in a broadcast network is in the forward direction only. This prevents multiple slaves from simultaneously send data to a single master. This is different to a shared channel, which also has a single master and multiple slaves; however, there is an addressing scheme that allows for data flow in both directions (refer next section).

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

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

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 3-1.

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 Data Payload

Shared Address LSB	Shared Address MSB	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
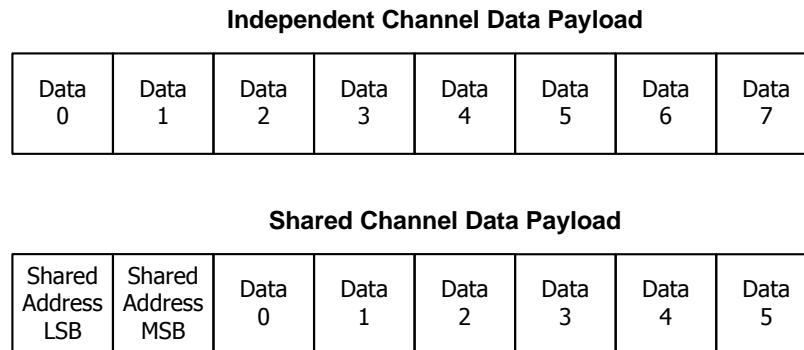
Figure 5-4. Independent and 2-byte Shared Channel Data Payloads.

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.

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 a shared channel, 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 rate. 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 shared channel is shown in Figure 5-5, with master node M, and four slave nodes addressed 1:4.

Shared Address Field:
[01][00]

Shared Address Field:
[04][00]

Shared Address Field:
[00][00]

(a)   (b)   (c)
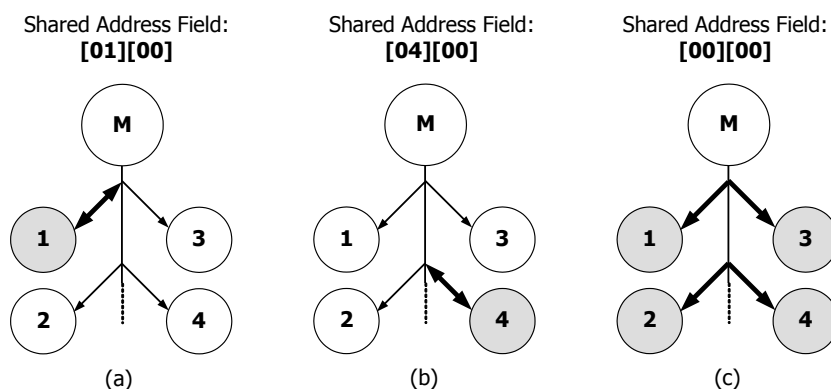
Figure 5-5. Example Shared Channel. Grey nodes indicate the node's host received data from ANT. Arrows indicate direction of data flow.

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 node's 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 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 have 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 (~18 mA) and 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 (see 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.

Please note, not all modules can support continuous scanning modes; refer to the datasheets 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, 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 as, 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 below.

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.
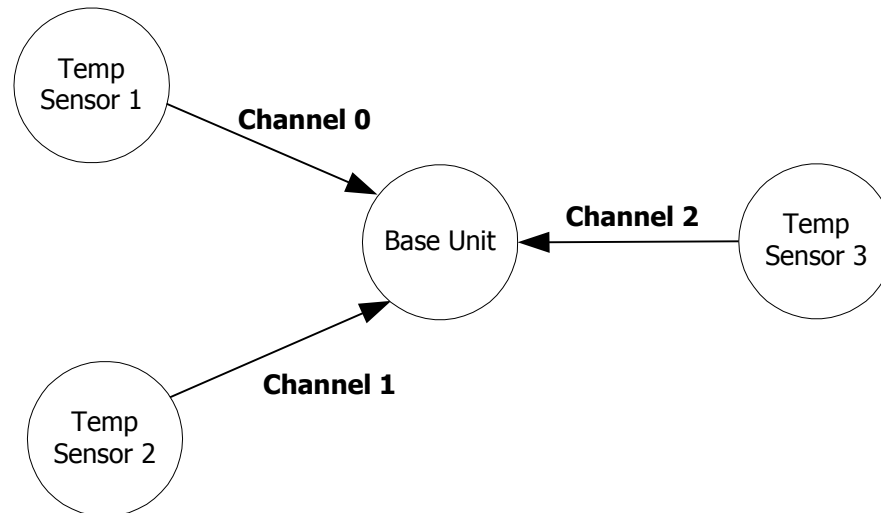
Figure 6-1. Example ANT Network for use in Device Pairing

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. Begins 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 (check datasheets for capabilities) is the inclusion/exclusion list. Up to four known channel IDs can be sent to the module and stored in this list. All fields must be defined and contain non-zero values. When enabled and configured as an inclusion list, the channel ID's stored will be the only channel IDs accepted in a wild card search. These means that the slave will only connect to one of the specific master channel ID's listed. Similarly, if this feature is configured as an exclusion list, the slave will not acquire any master with a listed channel ID.

Refer to sections 9.5.2.9, 9.5.2.10 and "Device Pairing" application note for more details.

6.3 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 set its device ID to search for a specific device type (say heart rate monitor), but placed a wildcard in all other fields, and there were four heart rate monitors in range (Figure 6-2 a, grey shading indicating slave's range); then, on opening its channel it could pair to any one of the four heart rate monitors depending on which transmitting master it found first.

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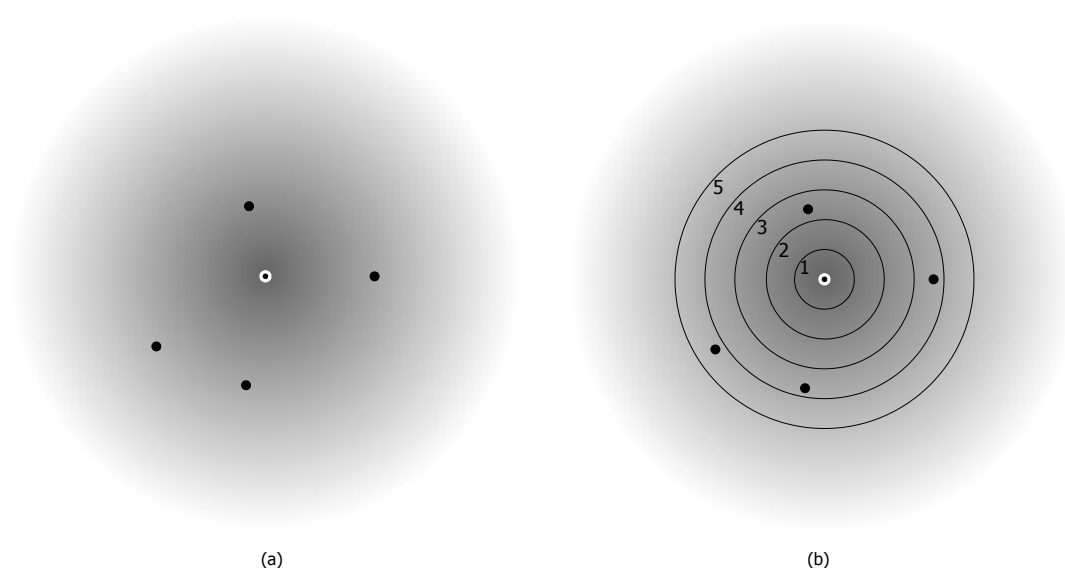
Figure 6-2. (a) Standard search (b) Proximity search, showing bins 1-5 (of maximum 10).

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 connecting 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).

(a)   (b)   (c)
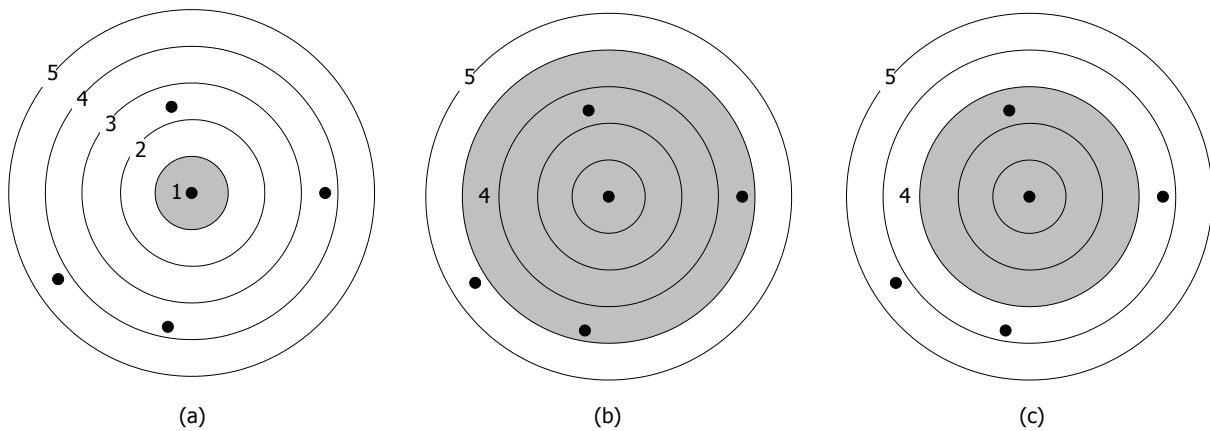
Figure 6-3. Varying Proximity Thresholds.

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

The 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 datasheets for capabilities.

7   ANT Interface

The host application and ANT 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.

7.1   Message Structure

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

Sync	Msg Length	Msg ID	Data_1	Data_2	. . .	Data_N	Check sum
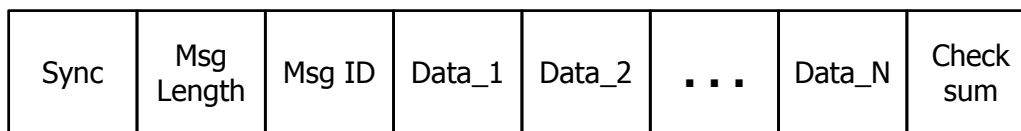
Figure 7-1. ANT Serial Message Structure

As shown above, each message begins with a SYNC byte and ends with a CHECKSUM. The bytes are sent lsb first. The following table describes each component of the serial message shown above.

Byte #	Name	Length	Description
0	SYNC	1 Byte	Fixed value of 10100100 or 10100101 (msb:lsb)
1	MSG LENGTH	1 Byte	Number of data bytes in the message. 1 < N < Max_Data_Size
2	MSG ID	1 Byte	Data Type Identifier 0: Invalid 1..255: Valid Data Type (See Section 9 for details)
3..N+2	DATA_1..DATA_N	N Bytes	Data bytes
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 both types of ANT interfaces: Host MCU ⇔ ANT and Host PC Interface ⇔ ANT. Message formatting is first presented in summary form, which includes message length, ID and data fields of each respective message type.

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 Data1, and the most significant byte to Data2.

7.1.1   Extended Messages Format

Extended messages allow ANT to pass the channel ID information to the host, along with the received data message. There are two formats supported by ANT, flagged and legacy, depending on the ANT device type (refer to Section 9.4 and datasheets for capabilities). 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) is the same. However, instead of just

the 8-byte data payload (D0:D7). The host will now receive the 8-byte data (D0:D7) packet followed by a flag byte (0x80) indicating the presence of the channel ID bytes: device number, device type, and transmission type. 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 the channel ID bytes are present.

Standard Data Packet

Sync	ML	ID	C#	D0	D1	D2	D3	D4	D5	D6	D7	CS

Extended Info

Sync	Msg Length	Msg ID	Channel Number	Data 0	**. . .**	Data 7	Flag Byte	Device Number	Device Type	Trans' Type	Check sum

Flagged Extended Data Packet

Sync	Msg Length	Msg ID	Channel Number	Device Number	Device Type	Trans' Type	Data 0	**. . .**	Data 7	Check sum

Legacy Extended Data Packet Format
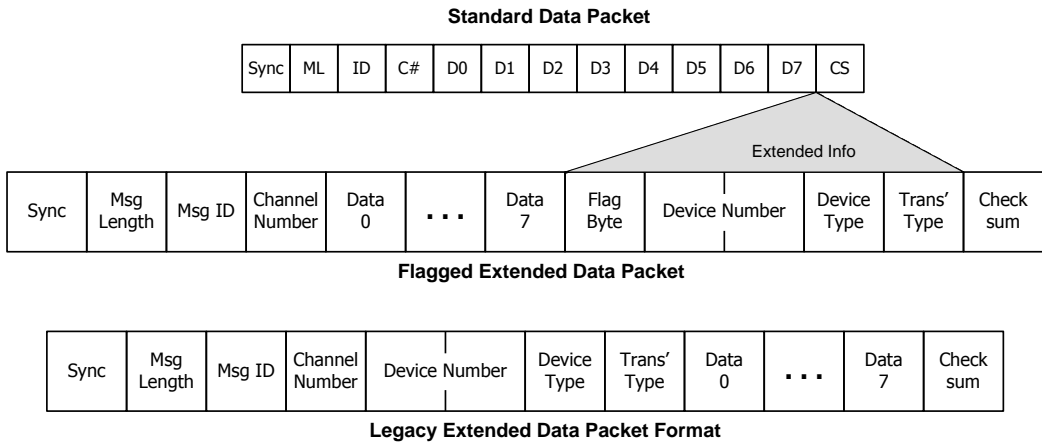
Figure 7-2. Extended data current and legacy formats.

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).

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.

8   Example ANT Network Implementation

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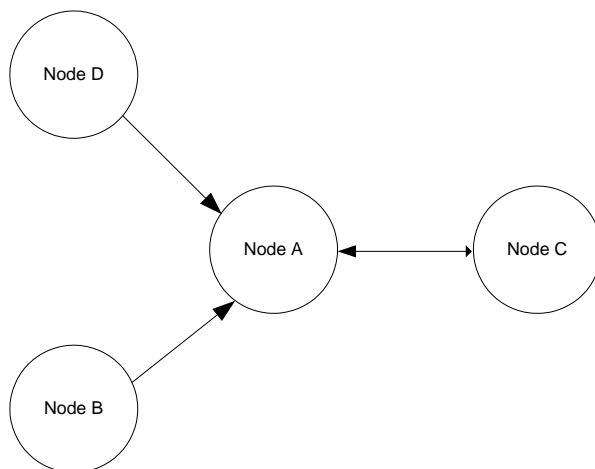
Figure 8-1. Example ANT Network for implementation

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.

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, device 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 configuration for each channel is shown in the following tables.

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

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
	Channel Type	0x00	Bidirectional Receive Channel
	Channel Period	8192	Default 4Hz Message Rate
	Data Type	0x4F	Acknowledged

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.

NODE B (Master)

Assign Channel
Channel #: 0
Type: 0x10
Network #: 0

Set Channel ID
Channel #:0
Device #: 1
Device type: 1
Trans' type: 1

Open Channel

Transmit Messages
Send Broadcast Data (0x4E)

NODE A (Slave)

Assign Channel
Channel #: 0
Type: 0x00
Network #:0

Set Channel ID
Channel #: 0
Device #: 1
Device type: 1
Trans' type: 1

Open Channel

Search
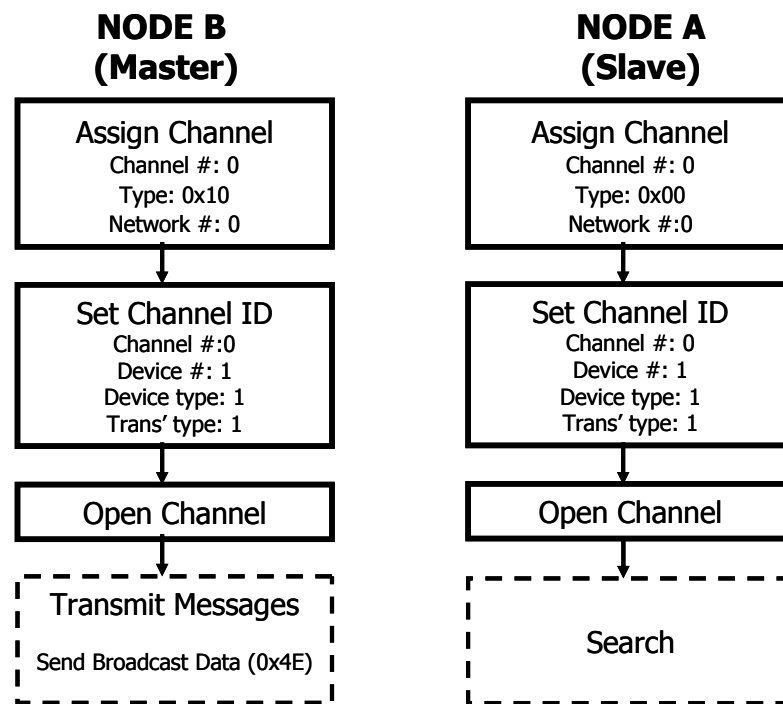
Figure 8-2. Node A & B Channel Establishment

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.

NODE C
(Master)

Assign Channel
Channel #: 0
Type: 0x10
Network #: 0

Set Channel ID
Channel #:0
Device #: 10
Device type: 2
Trans' type: 1

Open Channel

Transmit Messages

Send Acknowledged Data (0x4F)

NODE A
(Slave)

Assign Channel
Channel #: 1
Type: 0x00
Network #:0

Set Channel ID
Channel #: 1
Device #: 10
Device type: 2
Trans' type: 1

Open Channel

Search
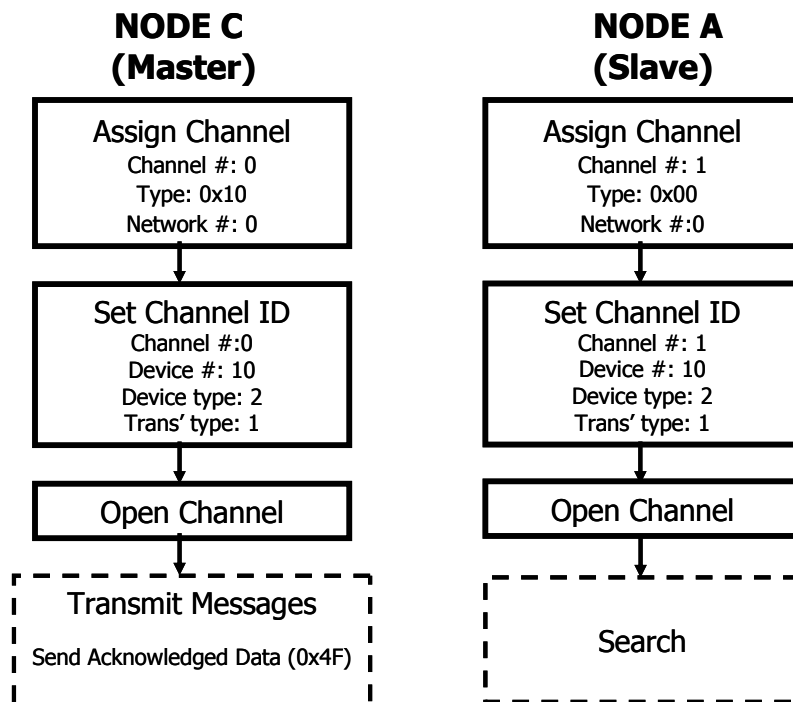
Figure 8-3. Node C & A Channel Establishment

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 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 implementation 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 to 6 bytes per packet.

As mentioned in the shared channel section (5.6), the central receiving node will be configured as master of the shared channel with the remaining nodes its slaves. Each slave will have a unique, two-byte shared channel address which will be known only to itself and the master. The updated network diagram for this setup is shown below.
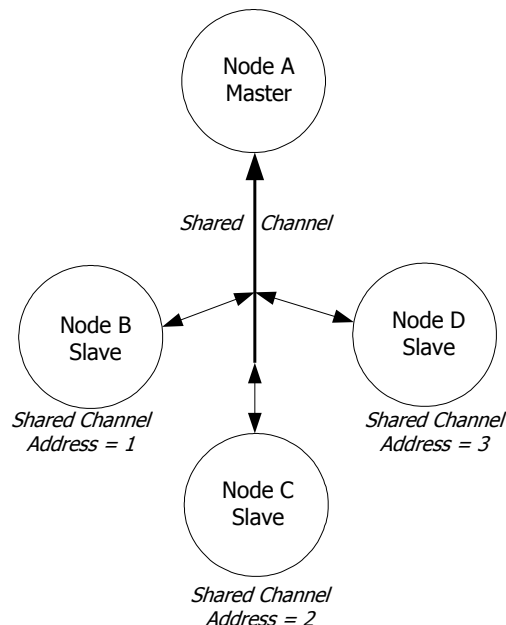
Figure 8-4. Shared channel implementation of sample network

Each node's channel configuration is shown below.

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, 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.

NODE A (Master)

Assign Channel
Channel #: 0
Type: 0x30
Network #: 0

Set Channel ID
Channel #:0
Device #: 3
Device type: 3
Trans' type: 3

Set Channel Period
Channel #:0
Message Period: 2371 (~12Hz)

Open Channel

Transmit Messages

Send Broadcast Data
[01][00][xx]…[xx]

Send Broadcast Data
[02][00][xx]…[xx]

Send Broadcast Data
[03][00][xx]…[xx]

NODE B (Slave)

Assign Channel
Channel #: 0
Type: 0x20
Network #:0

Set Channel ID
Channel #: 0
Device #: 3
Device type: 3
Trans' type: 3

Set Channel Period
Channel #:0
Message Period: 2371 (~12Hz)

Open Channel

Send Broadcast Data
[01][00][D0]…[D5]

Configures ANT that node has
Shared Address: [01][00]

Receive messages with Shared
Address: [01][00] or [00][00]

Send Broadcast Data
[01][00][D1]…[D5]

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 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.

**NODE C
(Slave)**

**NODE D
(Slave)**

Assign Channel
Channel #: 0
Type: 0x20
Network #:0

Assign Channel
Channel #: 0
Type: 0x20
Network #:0

Set Channel ID
Channel #: 0
Device #: 3
Device type: 3
Trans' type: 3

Set Channel ID
Channel #: 0
Device #: 3
Device type: 3
Trans' type: 3

Set Channel Period
Channel #:0
Message Period: 2371 (~12Hz)

Set Channel Period
Channel #:0
Message Period: 2371 (~12Hz)

Open Channel

Open Channel

Send Broadcast Data
[02][00][D0]...[D5]

Configures ANT that node has
Shared Address: [02][00]

Send Broadcast Data
[03][00][D0]...[D5]

Configures ANT that node has
Shared Address: [03][00]

Receive messages with Shared
Address: [02][00] or [00][00]

Send Acknowledged Data
[02][00][D1]...[D5]

Receive messages with Shared
Address: [03][00] or [00][00]

Send Broadcast Data
[03][00][D1]...[D5]

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 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 nodes 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.

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.

9.1.1 Config 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 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.

9.1.3 Notifications

Notifications allow ANT to inform the host of startup conditions.

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 'Response' column in Section 9.3 indicates whether ANT will send a response message to the respective command.

9.3 ANT Message Summary

Class	Type	ANT PC Interface Function Refer Section #	Reply	From	Len	Msg ID	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9
Config. Messages	Unassign Channel	ANT_UnassignChannel() 9.5.2.1 (p46)	Yes	Host	1	0x41	Channel Number								
	Assign Channel	ANT_AssignChannel() 9.5.2.2 (p46)	Yes	Host	3	0x42	Channel Number	Channel Type	Network Number	[Extended Assign't]					
	Channel ID	ANT_SetChannelId() 9.5.2.3 (p47)	Yes	Host	5	0x51	Channel Number	Device number		Device Type ID	Trans. Type				
	Channel Period	ANT_SetChannelPeriod() 9.5.2.4 (p48)	Yes	Host	3	0x43	Channel Number	Messaging Period							
	Search Timeout	ANT_SetChannelSearchTimeout() 9.5.2.5 (p49)	Yes	Host	2	0x44	Channel Number	Search Timeout							
	Channel RF Frequency	ANT_SetChannelRFFreq() 9.5.2.6 (p50)	Yes	Host	2	0x45	Channel Number	RF Frequency							
	Set Network	ANT_SetNetworkKey() 9.5.2.7 (p50)	Yes	Host	9	0x46	Net #	Key 0	Key 1	Key 2	Key 3	Key 4	Key 5	Key 6	Key 7
	Transmit Power	ANT_SetTransmitPower() 9.5.2.8 (p51)	Yes	Host	2	0x47	0	TX Power							
	ID List Add	ANT_AddChannelID() 9.5.2.9 (p51)	Yes	Host	6	0x59	Channel Number	Device number		Device Type ID	Trans. Type	List Index			
	ID List Config	ANT_ConfigList() 9.5.2.10 (p52)	Yes	Host	3	0x5A	Channel Number	List Size	Exclude						
	Channel Transmit Power	ANT_SetChannelTxPower() 9.5.2.11 (p52)	Yes	Host	2	0x60	Channel Number	TX Power							
	Low Priority Search Timeout	ANT_SetLowPriorityChannelSearch Timeout() 9.5.2.12 (p52)	Yes	Host	2	0x63	Channel Number	Search Timeout							
	Serial Number Set Channel ID	ANT_SetSerialNumChannelId() 9.5.2.13 (p53)	Yes	Host	3	0x65	Channel Number	Device Type ID	Trans. Type						
	Enable Ext RX Mesgs	ANT_RxExtMesgsEnable() 9.5.2.14 (p54)	Yes	Host	2	0x66	0	Enable							
	Enable LED	ANT_EnableLED() 9.5.2.15 (p54)	Yes	Host	2	0x68	0	Enable							
	Crystal Enable	ANT_CrystalEnable() 9.5.2.16 (p54)	Yes	Host	1	0x6D	0								
	Frequency Agility	ANT_ConfigFrequencyAgility() 9.5.2.17 (p55)	Yes	Host	4	0x70	Channel Number	Freq' 1	Freq' 2	Freq' 3					
	Proximity Search	ANT_SetProximitySearch() 9.5.2.18 (p55)	Yes	Host	2	0x71	Channel Number	Search Threshold							
	Set USB Info	ANT_SetUSBDescriptorString() 9.5.2.19 (p56)	Yes	Host			Description String (refer section 9.5.2.19)								

Class	Type	ANT PC Interface Function Refer Section #	Reply	From	Len	Msg ID	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9
Notifications	Startup Message	->ResponseFunc(-, 0x6F) 9.5.3.1 (p57)	-	ANT	1	0x6F	Startup Message								
Control Messages	SystemReset	ANT_ResetSystem() 9.5.4.1 (p57)	No	Host	1	0x4A	0								
	Open Channel	ANT_OpenChannel() 9.5.4.2 (p57)	Yes	Host	1	0x4B	Channel Number								
	Close Channel	ANT_CloseChannel() 9.5.4.3 (p58)	Yes	Host	1	0x4C	Channel Number								
	Open Rx Scan Mode	ANT_OpenRxScanMode() 9.5.4.5 (p58)	Yes	Host	1	0x5B	0								
	Request Message	ANT_RequestMessage() 9.5.4.4 (p58)	Yes	Host	2	0x4D	Channel Number	Message ID							
	Sleep Message	ANT_SleepMessage() 9.5.4.6 (p59)	No	Host	1	0xC5	0								
Data Messages	Broadcast Data	ANT_SendBroadcastData() ->ChannelEventFunc(Chan, EV) 9.5.5.1 (p59)	No	Host/ ANT	9	0x4E	Channel Number	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
	Acknowledge Data	ANT_SendAcknowledgedData() ->ChannelEventFunc(Chan, EV) 9.5.5.2 (p63)	No	Host/ ANT	9	0x4F	Channel Number	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
	Burst Transfer Data	ANT_SendBurstTransferPacket() ->ChannelEventFunc(Chan, EV) 9.5.5.3 (p67)	No	Host/ ANT	9	0x50	Sequence /Channel Number	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
Channel Event Messages	Channel Response / Event	->ChannelEventFunc(Chan, MessageCode) or ->ResponseFunc(Chan, MsgID) 9.5.6.1 (p73)	-	ANT	3	0x40	Channel Number	Message ID	Message Code						
Requested Response Messages	Channel Status	->ResponseFunc(Chan,0x52) 9.5.7.1 (p76)	-	ANT	2	0x52	Channel Number	Channel Status							
	Channel ID	->ResponseFunc(Chan,0x51) 9.5.7.2 (p77)	-	ANT	5	0x51	Channel Number	Device number		Device Type ID	Man ID				
	ANT Version	->ResponseFunc(-, 0x3E) 9.5.7.3 (p77)	-	ANT	11	0x3E	Ver0	Ver1	Ver2	Ver3 / Ver4	Ver5 / Ver6	Ver7	Ver8	Ver9	Ver10
	Capabilities	->ResponseFunc(-, 0x54) 9.5.7.4 (p77)	-	ANT	6	0x54	Max Channels	Max Networks	Standard Options	Advanced Options	Adv' Options 2	Rsvd			
	Serial Number	->ResponseFunc(-, 0x61) 9.5.7.5 (p78)	-	ANT	4	0x61	Serial Number								
Test Mode	CW Init	ANT_InitCWTestMode() 9.5.8.1 (p79)	Yes	Host	1	0x53	0								
	CW Test	ANT_SetCWTestMode() 9.5.8.2 (p79)	Yes	Host	3	0x48	0	TX Power	RF Freq						

Class	Type	ANT PC Interface Function	Reply	From	Len	Msg ID	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10	Data 11	Data 12	Data 13
Ext Data Messages	Extended Broadcast Data[¥]	ANT_SendExtBroadcastData()[¥] ->ChannelEventFunc(Chan, EV) 9.5.9.1 (p80)	No	Host/ ANT[¥]	13	0x5D	Channel Number	Device number		Device Type ID	Trans. Type	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7
	Extended Acknowledged Data[¥]	ANT_SendExtAcknowledgedData() ->ChannelEventFunc(Chan, EV) 9.5.9.2 (p81)	No	Host/ ANT[¥]	13	0x5E	Channel Number	Device number		Device Type ID	Trans. Type	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7
	Extended Burst Data[¥]	ANT_SendExtBurstTransferPacket( ) ->ChannelEventFunc(Chan, EV) 9.5.9.3 (p82)	No	Host/ ANT[¥]	13	0x5F	Sequence /Channel Number	Device number		Device Type ID	Trans. Type	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7

[¥]   These are legacy formats for 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	ANT PC Interface Function	nRF24AP1 and AP1 Modules	ANT11TRx1 Chipsets & modules	AT3 Chipsets & modules	nRF24AP2 & AP2 Modules	nRF24AP2-USB
Config. Messages	Unassign Channel	ANT_UnAssignChannel()	Yes	Yes	Yes	Yes	Yes
	Assign Channel	ANT_AssignChannel()	Yes (3-Bytes)	Yes (3-Bytes)	Yes (3-Bytes)	Yes (3or4-Bytes)	Yes (3or4-Bytes)
	Channel ID	ANT_SetChannelId()	Yes	Yes	Yes	Yes	Yes
	Channel Period	ANT_SetChannelPeriod()	Yes	Yes	Yes	Yes	Yes
	Search Timeout	ANT_SetChannelSearchTimeout()	Yes	Yes	Yes	Yes	Yes
	Channel RF Frequency	ANT_SetChannelRFFreq()	Yes	Yes	Yes	Yes	Yes
	Set Network	ANT_SetNetworkKey()	Yes	Yes	Yes	Yes	Yes
	Transmit Power	ANT_SetTransmitPower()	Yes	Yes	Yes	Yes	Yes
	ID List Add	ANT_AddChannelID()	No	No	Yes	Yes	Yes
	ID List Config	ANT_ConfigList()	No	No	Yes	Yes	Yes
	Channel Transmit Power	ANT_SetChannelTxPower()	No	No	Yes	Yes	Yes
	Low Priority Search Timeout	ANT_SetLowPriorityChannelSearchTimeout()	No	No	Yes	Yes	Yes
	Serial Number Set Channel ID	ANT_SetSerialNumChannelId()	No	No	Yes	No	No
	Enable Ext RX Mesgs	ANT_RxExtMesgsEnable()	No	No	Yes	Yes	Yes
	Enable LED	ANT_EnableLED()	No	No	Yes	No	No
	Crystal Enable	ANT_CrystalEnable()	No	No	No	Yes	No
	Frequency Agility	ANT_ConfigFrequencyAgility()	No	No	No	Yes	Yes
	Proximity Search	ANT_SetProximitySearch()	No	No	No	Yes	Yes
	Set USB Info	ANT_SetUSBDescriptorString()	No	No	No	No	Yes
Notifications	Startup Message	->ResponseFunc(-,0xC5)	No	No	No	Yes	Yes
Control Messages	SystemReset	ANT_ResetSystem()	Yes	Yes	Yes	Yes	Yes
	Open Channel	ANT_OpenChannel()	Yes	Yes	Yes	Yes	Yes
	Close Channel	ANT_CloseChannel()	Yes	Yes	Yes	Yes	Yes
	Open Rx Scan Mode	ANT_OpenRxScanMode()	No	No	Yes	Yes	Yes
	Request Message	ANT_RequestMessage()	Yes	Yes	Yes	Yes	Yes
	Sleep Message	ANT_Sleep()	No	No	No	Yes	Yes
Data Messages	Broadcast Data	ANT_SendBroadcastData() ->ChannelEventFunc(Chan, EV)	Yes	Yes	Yes	Yes	Yes

	Acknowledge Data	ANT_SendAcknowledgedData() ->ChannelEventFunc(Chan, EV)	Yes	Yes	Yes	Yes	Yes
	Burst Transfer Data	ANT_SendBurstTransferPacket() ->ChannelEventFunc(Chan, EV)	Yes	Yes	Yes	Yes	Yes
Channel Event Messages	Channel Response / Event	->ChannelEventFunc(Chan, MessageCode) or ->ResponseFunc(Chan, MsgID);	Yes	Yes	Yes	Yes	Yes
Requested Response Messages	Channel Status	->ResponseFunc(Chan,0x52)	Yes	Yes	Yes	Yes	Yes
	Channel ID	->ResponseFunc(Chan,0x51)	Yes	Yes	Yes	Yes	Yes
	ANT Version	->ResponseFunc(-, 0x3E)	No	Yes	Yes	Yes	Yes
	Capabilities	->ResponseFunc(-, 0x54)	Yes (4–bytes)	Yes (4–bytes)	Yes (6–bytes)	Yes (6–bytes)	Yes (6–bytes)
	Serial Number	->ResponseFunc(-, 0x61)	No	No	Yes	No	No
Test Mode	CW Init	ANT_InitCWTestMode()	Yes	Yes	Yes	Yes	Yes
	CW Test	ANT_SetCWTestMode()	Yes	Yes	Yes	Yes	Yes
Ext Data Messages	Extended Broadcast Data	ANT_SendExtBroadcastData() ->ChannelEventFunc(Chan, EV)	No	No	Yes	Yes	Yes
	Extended Acknowledge Data	ANT_SendExtAcknowledgedData() ->ChannelEventFunc(Chan, EV)	No	No	Yes	Yes	Yes
	Extended Burst Data	ANT_SendExtBurstTransferPacket() ->ChannelEventFunc(Chan, EV)	No	No	Yes	Yes	Yes

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	nRF24AP2-USB
RESPONSE_NO_ERROR	Yes	Yes	Yes	Yes	Yes
EVENT_RX_SEARCH_TIMEOUT	Yes	Yes	Yes	Yes	Yes
EVENT_RX_FAIL	Yes	Yes	Yes	Yes	Yes
EVENT_TX	Yes	Yes	Yes	Yes	Yes
EVENT_TRANSFER_RX_FAILED	Yes	Yes	Yes	Yes	Yes
EVENT_TRANSFER_TX_COMPLETED	Yes	Yes	Yes	Yes	Yes
EVENT_TRANSFER_TX_FAILED	Yes	Yes	Yes	Yes	Yes
EVENT_CHANNEL_CLOSED	Yes	Yes	Yes	Yes	Yes
EVENT_RX_FAIL_GO_TO_SEARCH	No	Yes	Yes	Yes	Yes
EVENT_CHANNEL_COLLISION	No	Yes	Yes	Yes	Yes
EVENT_TRANSFER_TX_START	No	No	Yes	Yes	Yes
CHANNEL_IN_WRONG_STATE	Yes	Yes	Yes	Yes	Yes
CHANNEL_NOT_OPENED	Yes	Yes	Yes	Yes	Yes
CHANNEL_ID_NOT_SET	Yes	Yes	Yes	Yes	Yes
CLOSE_ALL_CHANNELS	No	No	Yes	Yes	Yes
TRANSFER_IN_PROGRESS	Yes	Yes	Yes	Yes	Yes
TRANSFER_SEQUENCE_NUMBER_ERROR	Yes	Yes	Yes	Yes	Yes
TRANSFER_IN_ERROR	No	No	Yes	Yes	Yes
INVALID_MESSAGE	Yes	Yes	Yes	Yes	Yes
INVALID_NETWORK_NUMBER	Yes	Yes	Yes	Yes	Yes
INVALID_LIST_ID	No	No	Yes	Yes	Yes
INVALID_SCAN_TX_CHANNEL	No	No	Yes	Yes	Yes
INVALID_PARAMETER_PROVIDED	No	No	No	Yes	Yes
EVENT_QUE_OVERFLOW	No	No	No	Yes	Yes
NVM_FULL_ERROR	No	No	Yes	No	No
NVM_WRITE_ERROR	No	No	Yes	No	No
USB_STRING_WRITE_FAIL	No	No	No	No	Yes

9.5   ANT Message Details

This section provides detailed information regarding ANT message and data fields for each ANT message type.

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).

These values can be determined for the specific ANT implementation by requesting the capability message (see Section 0).

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
Network Number	UCHAR	0..MAX_NET-1	Specifies the network address to be used for this channel. Set this to 0, to use the default public network. See Network Address for more details.
Extended Assignment [optional]	UCHAR	As specified	0x01 – Background Scanning Channel Enable 0x04 – Frequency Agility 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_AssignChannelExtl(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 and background scanning channel. For more information on these features see sections 5.2.1.4.1, 5.2.1.4.2, 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 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);
  // wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 0, 120, 1); // device number is wild-card
/**************************************************/
// Pairing bit on Rx channel
ANT_AssignChannel(0, 0x00, 0);
  // wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 0, 248, 1); // device number is wild-card, 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.

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 is irrelevant.

Note that Transmission Type and Device Type IDs are assigned and regulated to maintain network integrity and interoperability, except for the free default 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:

1. A smaller device period increases the message rate and thus increases system power consumption (see respective ANT product datasheet for details).

2. A smaller device period (faster message rate) allows higher Broadcast data-transfer rates.

3. 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 select 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 (25seconds) AP1: 12 (30 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 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.

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 unassigned.
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 0	UCHAR	0..255	Network byte 0
Network Key 1	UCHAR	0..255	Network byte 1
Network Key 2	UCHAR	0..255	Network byte 2
Network Key 3	UCHAR	0..255	Network byte 3
Network Key 4	UCHAR	0..255	Network byte 4
Network Key 5	UCHAR	0..255	Network byte 5
Network Key 6	UCHAR	0..255	Network byte 6
Network Key 7	UCHAR	0..255	Network byte 7

```
// 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 uninitialised. For non-AP1 devices, all remaining network numbers default to the public network.

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 have 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 free default 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..3	3 (0dBm)	0 = TX Power -20 dBm 1 = TX Power -10 dBm 2 = TX Power -5 dBm 3 = TX Power 0 dBm

```
// 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 (0x59)
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. Must not contain a wildcard.
Device Type ID	UCHAR (7bits)	0-6	0..127	The device type. Must not contain a wildcard value.
Transmission Type	UCHAR	-	0..255	The transmission type. Must not contain a wildcard value.
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, check datasheets for capabilities. This message is sent to the module to add channel IDs to the inclusion/exclusion list. When this list is used, these ID's will either be the only IDs accepted in a wild card search or ID's that will not be discovered at all. The use of these ID's is enabled by the ConfigList command detailed below. A maximum of 4 IDs can be placed in the list.

9.5.2.10 Config List ID (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 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, check datasheets for capabilities. This message is sent to ANT to configure the inclusion/exclusion list. The size determines which ID's in the list are to be used (setting a size of 0 disables the include/exclude list) and the exclude variable determines whether the IDs are to be found or to be ignored when the device is searching.

9.5.2.11 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..3	0 = TX Power -20 dBm 1 = TX Power -10 dBm 2 = TX Power -5 dBm 3 = TX Power 0 dBm

```
// 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, check datasheets for capabilities.

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.

9.5.2.12 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 is 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 AN11 ANT Channel Search application note for more details.

9.5.2.13 Serial Number 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);
/****************************************************/
// Rx channel
ANT_AssignChannel(0, 0x00, 0);
  // wait for RESPONSE_NO_ERROR
ANT_SetSerialNumChannelId(0, 120, 123); // device number is wild-card
/****************************************************/
// Pairing bit on Rx channel
ANT_AssignChannel(0, 0x00, 0);
  // wait for RESPONSE_NO_ERROR
ANT_SetSerialNumChannelId(0, 248, 123); // device number is wild-card, device type 120 with pairing bit ON
```

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

9.5.2.14   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, check datasheets for capabilities. 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. See section 7.1.1 for more information regarding the extended data bytes.

9.5.2.15   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, check datasheets for capabilities. 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 a RF transmit or receive event is detected by the module.

9.5.2.16   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 startup 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.

9.5.2.17 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, check datasheets for capabilities. 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 (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.

9.5.2.18 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, check datasheets for capabilities. 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.

9.5.2.19   Set USB Descriptor String (0xC7)

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

Parameters	Type	Range	Description
ucStringNum	UCHAR	0..3	Descriptor String Number 0 – PID/VID 1 – Manufacturer String 2 – Device String 3 – Serial Number String
pucDescString[0]	UCHAR	0..255	String Character 0/VID LSB
pucDescString[1]	UCHAR	0..255	String Character 1/VID MSB
pucDescString[2]	UCHAR	0..255	String Character 2/PID LSB
pucDescString[3]	UCHAR	0..255	String Character 3/PID MSB
pucDescString[n]	UCHAR	0..255	String Character n
pucDescString[ucStringSize -1]	UCHAR	0	NULL character (except for string 0)
ucStringSize	UCHAR	1..32	String Length

```
// Example Usage
UCHAR aucDescString0 = {0xFC, 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 (0, aucDescString2, sizeof(aucDescString2)); // set the Device String
ANT_SetUSBDescriptorString (0, 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   StartupMessage(0x6F)
ResponseFunc (-, 0x6F)

Please note this message is only available on specific devices, check datasheets for capabilities. The startup message returns a 1-byte bit field, on every ANT power up or reset event. The bitfield indicates the type of reset occurred.

Parameters	Type	Range	Description
Startup Message	UCHAR	0..255	The Startup Message bitfield 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

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 still 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);

Parameters	Type	Range	Description
Channel Number	UCHAR	0..MAX_CHAN-1	The channel number associated with the message request
Message ID Requested	UCHAR	See Section 9.3	ID of the message being requested

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

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, and Capabilities. 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();
```

This message is sent to the module to open 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 so no other channels can operate when the node is in continuous scanning mode. The node will pick up any message, regardless of period, that is being transmitted on its 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.

9.5.4.6 SleepMessage (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 sleep mode			

Please note this message is only available on specific devices, check datasheets for capabilities. The Sleep command will put ANT into an ultra-low 0.5uA 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 Chips and Modules document for more details.

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

9.5.5 Data Messages

There are three methods for sending and receiving data on a channel. These methods 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 the channel ID.

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 data messages must be enabled using the ANT_RxExtMesgsEnable (0x66) message.

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	0x80	0x80 – indicates presence of channel ID bytes
[Device Number]	USHORT (little endian)	0..65535	Optional extended messages bytes. Only included if flag byte indicates its presence
[Device Type]	UCHAR	0..255	Optional extended messages byte. Only included if flag byte indicates its presence
[Transmission Type]	UCHAR	0..255	Optional extended messages byte. Only included if flag byte indicates its presence

```
// 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 recieved 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
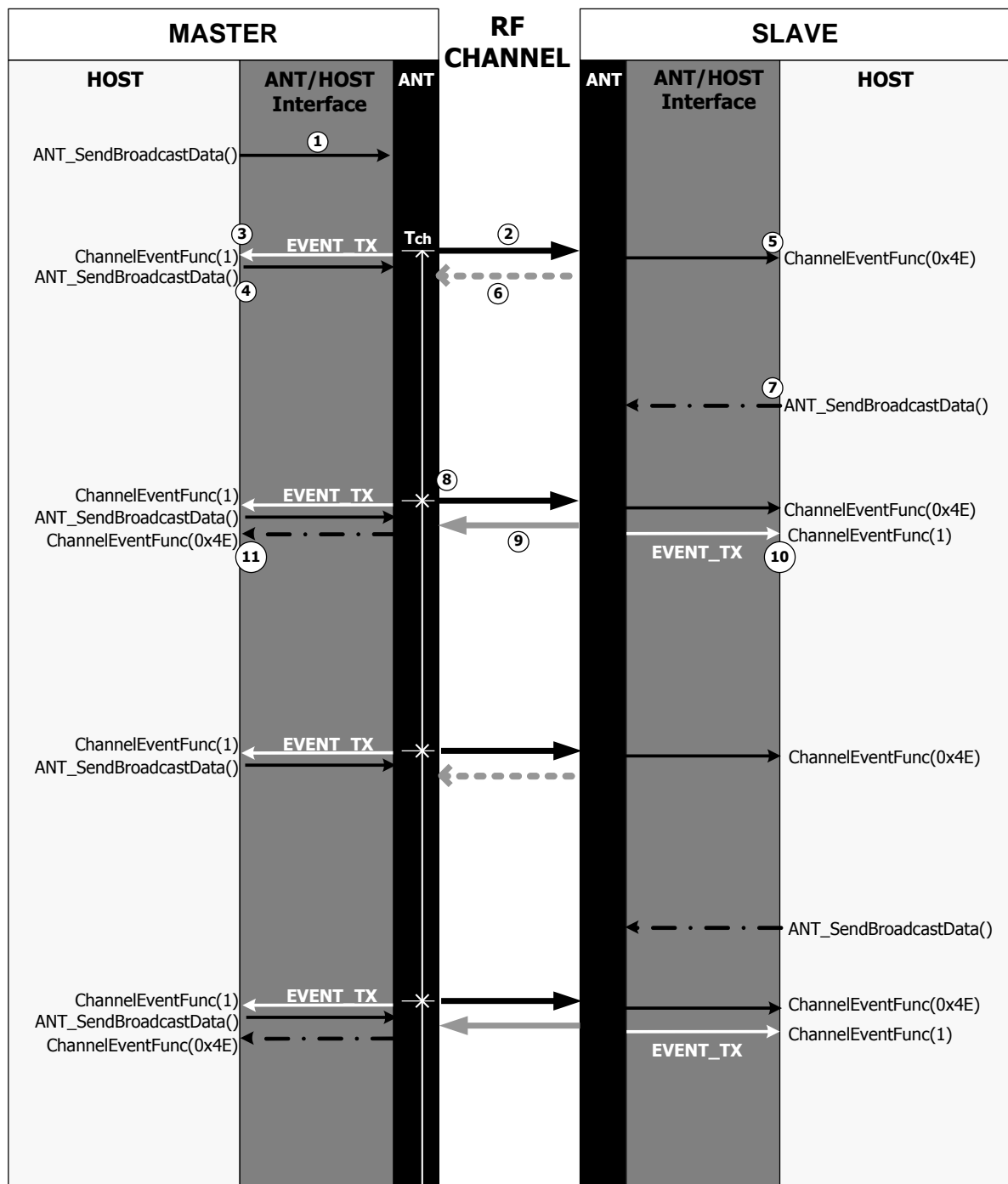
Master

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 buffers over the RF channel, master host receives an EVENT_TX, and slave 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 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 broadcast 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 the channel ID.

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_ACKNOWLEDGED for standard acknowledged messages and EVENT_RX_FLAG_ACKNOWLEDGED for flagged extended data messages.

Please note that flagged data messages must be enabled using the ANT_RxExtMesgsEnable (0x66) message.

Any application that processes flagged 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	0x80	0x80 – indicates presence of channel ID bytes
[Device Number]	USHORT (little endian)	0..65535	Optional extended messages bytes. Only included if flag byte indicates its presence
[Device Type]	UCHAR	0..255	Optional extended messages byte. Only included if flag byte indicates its presence
[Transmission Type]	UCHAR	0..255	Optional extended messages byte. Only included if flag byte indicates its presence

```
// 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 recieved 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 originated from channels configured for transmit only.

Figure 9-2 below describes the acknowledged message transactions from master host to ANT, over the RF channel to Slave ANT, host and vice versa in the reverse direction.
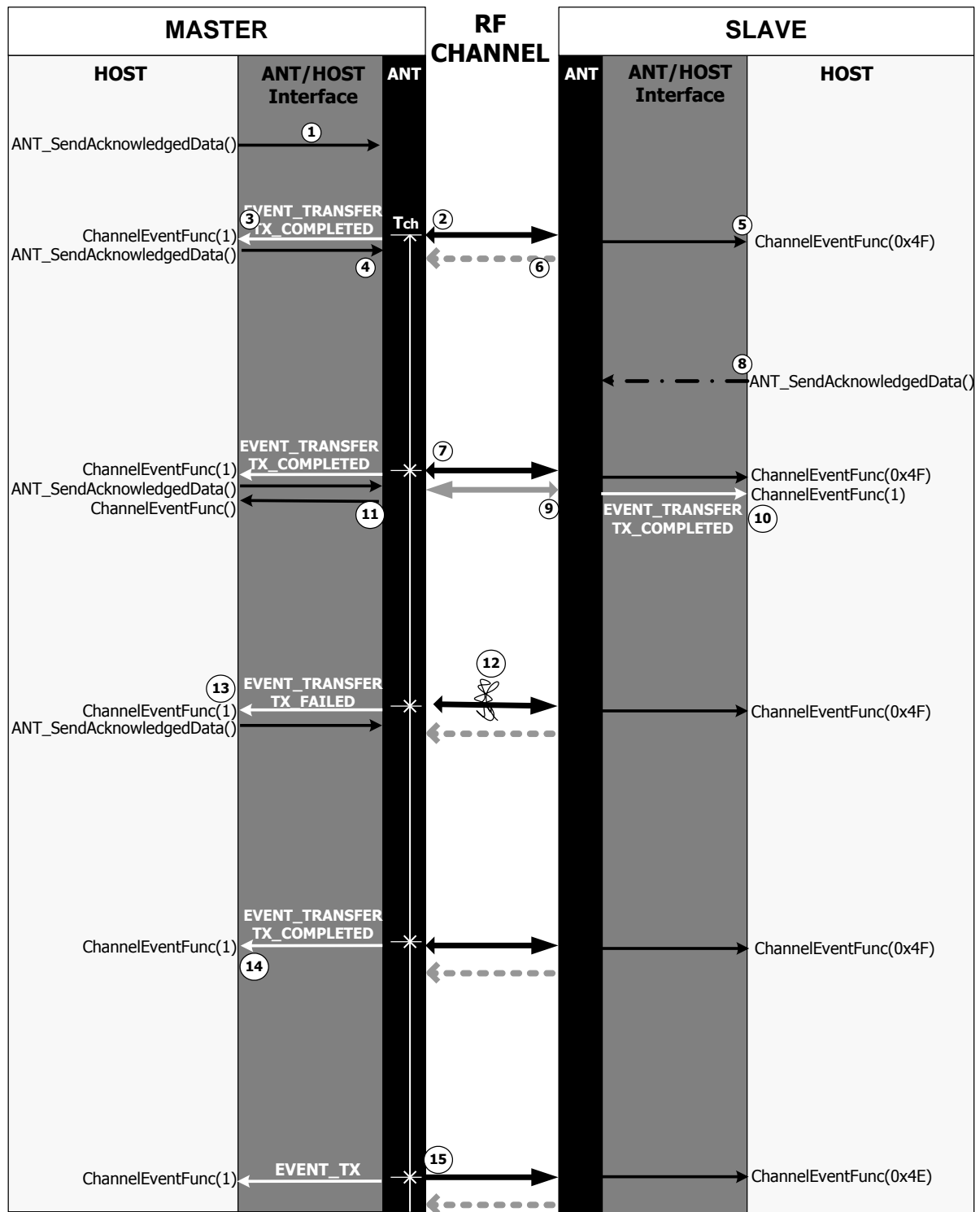
Figure 9-2 Acknowledged Data Sequence Diagram

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 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 broadcast 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 the channel ID. Subsequent message packets will not contain any extra messages and will be 9 bytes in length.

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 data messages must be enabled using the ANT_RxExtMesgsEnable (0x66) message.

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	0x80	0x80 – indicates presence of channel ID bytes Only present on 1st burst packet
[Device Number]	USHORT (little endian)	0..65535	Optional extended messages bytes. Only included if flag byte indicates its presence. Only present on 1st burst packet
[Device Type]	UCHAR	0..255	Optional extended messages byte. Only included if flag byte indicates its presence. Only present on 1st burst packet
[Transmission Type]	UCHAR	0..255	Optional extended messages byte. Only included if flag byte indicates its presence. Only present on 1st burst packet

```
// 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 recieved 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;
    }
  }
}
```

Figure 9-3 below describes the burst message transactions from master host to ANT, over the RF channel to Slave ANT, to host and back.
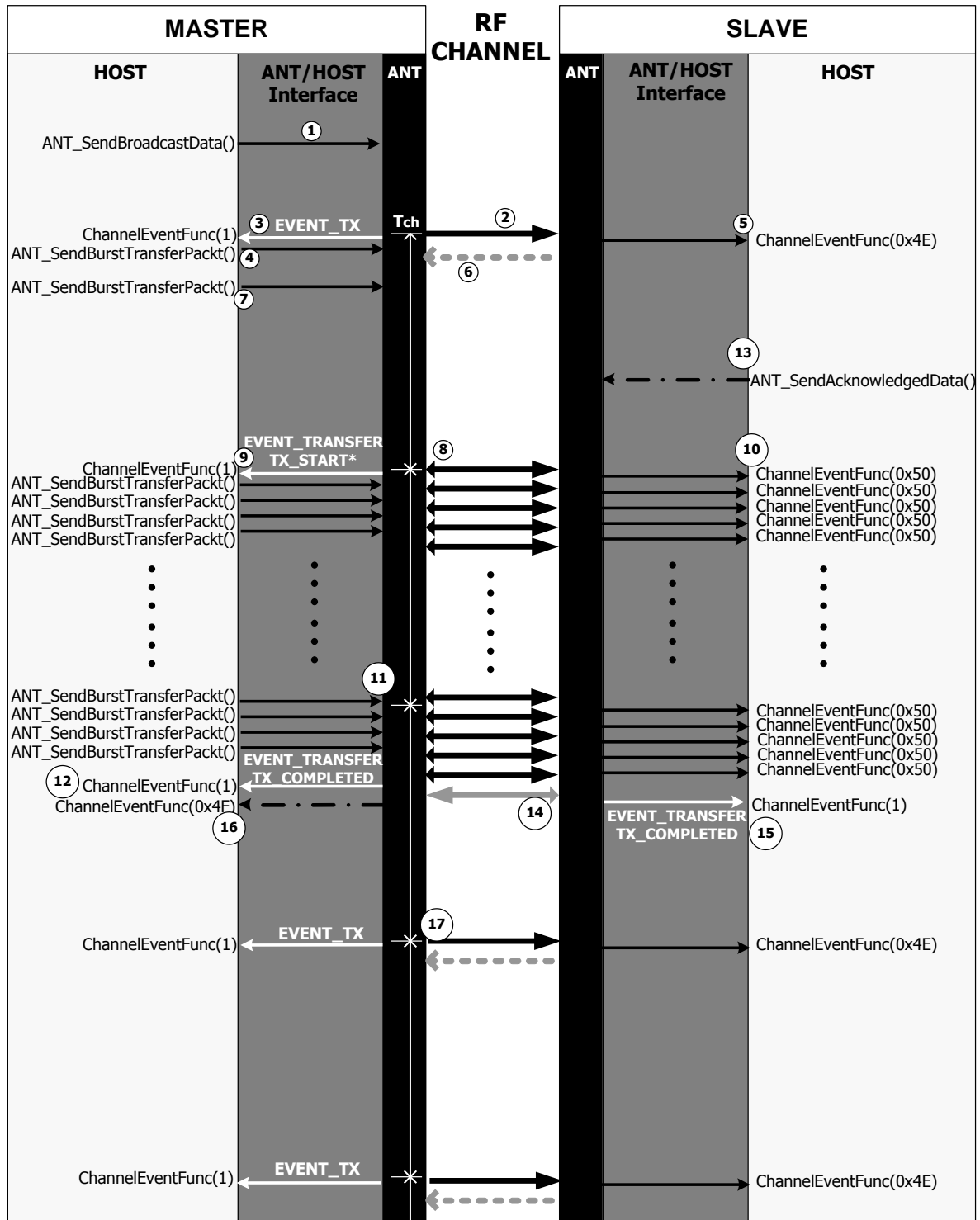

Figure 9-3 Burst Transfer Sequence Diagram

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 the transfer will be aborted and ANT will send an error message to the host 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 master host to ANT, over the RF channel to Slave ANT, to host and back. Also refer to the application note AN04 - Burst Transfers for more details.

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 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), 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 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 chips 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.

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.

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 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.

- Message code values are in decimal

Name	Value	Description
RESPONSE_NO_ERROR	0	Returned on a successful operation
EVENT_RX_SEARCH_TIMEOUT	1	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	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	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	A receive transfer has failed. This occurs when a Burst Transfer Message was incorrectly received.
EVENT_TRANSFER_TX_COMPLETED	5	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.
EVENT_TRANSFER_TX_FAILED	6	An Acknowledged Data message, or a Burst Transfer Message has been initiated and the transmission failed to complete successfully

Name	Value	Description
EVENT_CHANNEL_CLOSED	7	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	The channel has dropped to search mode after missing too many messages.
EVENT_CHANNEL_COLLISION	9	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	Sent after a burst transfer begins, effectively on the next channel period after the burst transfer message has been sent to the device.
CHANNEL_IN_WRONG_STATE	21	Returned on attempt to perform an action on a channel that is not valid for the channel's state
CHANNEL_NOT_OPENED	22	Attempted to transmit data on an unopened channel
CHANNEL_ID_NOT_SET	24	Returned on attempt to open a channel before setting a valid ID
CLOSE_ALL_CHANNELS	25	Returned when an OpenRxScanMode() command is sent while other channels are open.
TRANSFER_IN_PROGRESS	31	Returned on an attempt to communicate on a channel with a transmit transfer in progress.
TRANSFER_SEQUENCE_NUMBER_ERROR	32	Returned when sequence number is out of order on a Burst Transfer
TRANSFER_IN_ERROR	33	Returned when a burst message passes the sequence number check but will not be transmitted due to other reasons.
INVALID_MESSAGE	40	Returned when message has invalid parameters
INVALID_NETWORK_NUMBER	41	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	Returned when the provided list ID or size exceeds the limit.
INVALID_SCAN_TX_CHANNEL	49	Returned when attempting to transmit on ANT channel 0 in scan mode.
INVALID_PARAMETER_PROVIDED	51	Returned when invalid configuration commands are requested
EVENT_QUE_OVERFLOW	53	Only possible when using synchronous serial port. Indicates that one or more events was lost due to excessive latency in reading out events over the port.
NVM_FULL_ERROR	64	Returned when the NVM for SensRcore mode is full.
NVM_WRITE_ERROR	65	Returned when writing to the NVM for SensRcore mode fails.

```
// Example Usage
BOOL ANT_ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  switch (ucEvent)
  {
    case EVENT_RX_BROADCAST:
    {
      switch (ucChannel)
```

228 River Avenue, Cochrane, Alberta, Canada T4C 2C1

thisisant.com

Name	Value	Description

```
      {
        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;
    }
  }
}
```

9.5.7 Requested Response Messages

The following messages are returned in response to a Request Message (see 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)

This message returns the channel status information for the specified channel.

Parameters	Type	Range	Description
Channel Number	UCHAR	0.. MAX_CHAN-1	The channel number
Channel State	UCHAR (Bits 1:0)	0..3	State of the channel Un-Assigned = 0 Assigned = 1 Searching = 2 Tracking = 3
Reserved	UCHAR (Bits 7:2)	varies	Reserved

```
// Example Usage
BOOL ANT_ResponseFunction(UCHAR ucChannel, UCHAR ucResponseMesgID)
{
  Switch (ucResponseMesgID)
  {
    case MESG_CHANNEL_STATUS_ID:
    {
      switch (aucResponseBuffer[1]) // 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 to 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 channel and searching for this specific device. See pairing under the Usage section for more details.

Note that the Transmission Type and Device Type IDs are assigned and regulated to maintain network integrity, and interoperability, except for the free default 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 11-byte null-terminated version string, corresponding to the ANT host interface version.

Parameters	Type	Range	Description
Version Message	char[11]	1..255	9 byte string

Please note that this message is not supported on all ANT products.

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

Parameters	Type	Range	Description
Advanced Options	UCHAR	0..255	The Advanced Options bitfield is encoded as follows: Bit 1 - CAPABILLITES_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 - CAPABILLITES_SCRIPT_ENABLED Bit 7 - CAPABILLITES_SEARCH_LIST_ENABLED Other bits are reserved
Advanced Options 2 (available in new versions only)	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 Other bits are reserved
Reserved	UCHAR	varies	

9.5.7.5   Device Serial Number (0x61)
ResponseFunc (-, 0x61)

Please note this message is only available on specific devices, check datasheets for capabilities. The serial number is a 4-byte, little-endian, encoded unsigned integer. Please note that this message is not supported on all ANT products.

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

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..3	0 = TX Power -20 dB 1 = TX Power -10 dB 2 = TX Power -5 dB 3 = TX Power 0 dB
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 the 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 and 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 9.5.5 can be sent in the legacy extended data message format. These functions are now supported in nRF24AP2 as flagged extended message bytes in existing data messages. See Section 7.1.1 Extended Messages Format. However, AP2 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 for/from
Device Num	USHORT	0..65536	Device Number
Device Type	UCHAR	0..255	Device Type
Transmission Type	UCHAR	0..255	Transmission Type
Data 0	UCHAR	0..255	The first 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 an 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	Device Number
Device Type	UCHAR	0..255	Device Type
Transmission Type	UCHAR	0..255	Transmission Type
Data 0	UCHAR	0..255	The first 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	Device Number
Device Type	UCHAR	0..255	Device Type
Transmission Type	UCHAR	0..255	Transmission Type
Data 0	UCHAR	0..255	The first data byte
..			
Data 7	UCHAR	0..255	The eighth data byte

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

ANT
the power of less

thisisant.com

```
}
```

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 the www.thisisant.com website:

- 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.