ANT

Wireless Personal Area Network

ANT Message Protocol and Usage

Copyright Information and Usage Notice

Revision History

Revision	Effective Date	Description
1.0	April 9, 2005	Preliminary Version Release
1.31	April 29, 2005	External Release
1.32	June 1, 2005	DLL Information Added
1.33	June 20, 2005	Various updates to format, information layout and added examples of network implementations
1.34	September 28, 2005	Updated Appendix
1.35	November 18, 2005	Added Broadcast Address description to Shared Channels Section
1.36	November 23, 2005	Updated Appendix

PRELIMINARY

Table of Contents

1   Introduction

ANT is a 2.4GHz bidirectional wireless Personal Area Network (PAN) communications technology optimized for transferring low data-rate, low latency data between multiple ANT-enabled devices. The ultra-low power consumption of the ANT chipset guarantees an extended battery life even from low capacity supplies such as a coin cell battery, enabling use in heart rate monitors, bicycle computers, and wrist watches. The small size and low-cost implementation of ANT proves essential in allowing effortless integration into the tiny form factor of wrist watches, PDAs, and mobile phones.

ANT provides carefree handling of OSI layers including the Physical, Network and Transport 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.

Application / Presentation Layers	User Defined
Higher Level Security	
Network / Transport & Low Level Security	Implemented by ANT
Data Link Layer	
Physical Layer	
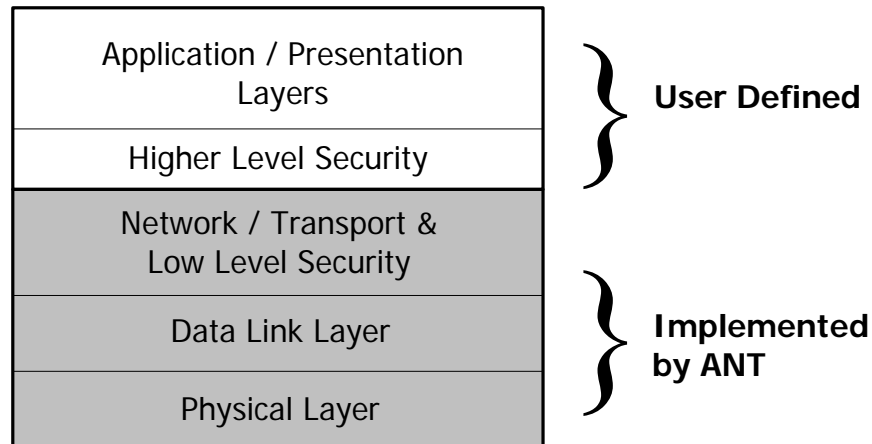
Figure 1-1: OSI Layer model of ANT

The interface between ANT and the Host application has been designed with the utmost simplicity in mind so that it 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. In addition a burst mode allows for the efficient transfer of large amounts of stored data to and from a PC or other computing device. The ANT system aggressively balances functionality, cost, size, and power consumption within the constraints of a mobile Personal Area Network. Typical applications include sensor integration, tagging systems, remote monitoring etc.

A typical ANT-enabled device consists of an application host MCU interfaced with an ANT module or chipset. 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 as well as provides examples of its usage.

2   The ANT Product Family

ANT technology has been incorporated into a family of products that allow 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 nRF24AP1 – A complete ANT implementation integrated into a single-chip RF protocol and transceiver Integrated Circuit (IC).

2. Two-chip ANT solutions combine an ANT-protocol MCU with a Nordic Semiconductor RF IC (such as the nRF2401A or nRF2402). A variety of two-chip options are available, providing either one-way or two-way RF communications, allowing the protocol to be scaled with the needs of the application. The two-chip solutions provide the ultimate in low power consumption.

ANT Module

A certification ready PCB module incorporates an ANT chipset and can be piggybacked onto an 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 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. This 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.
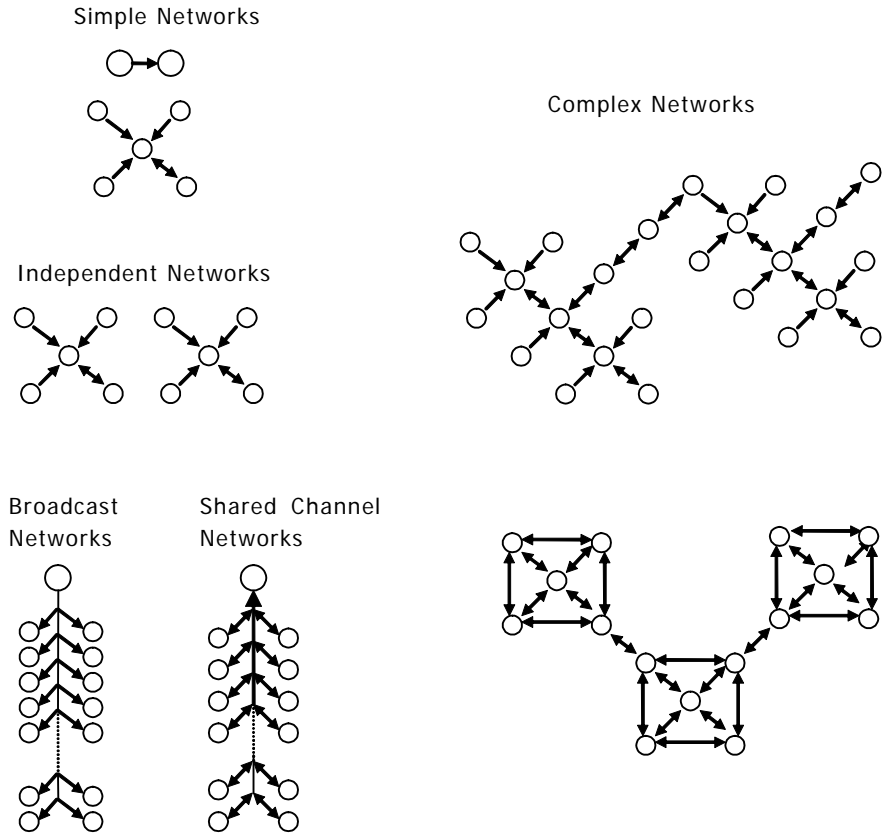
Figure 3-1: Example ANT Networks

For the purpose of illustration, a simple example is shown next to demonstrate the basic concept of ANT channels.

PRELIMINARY
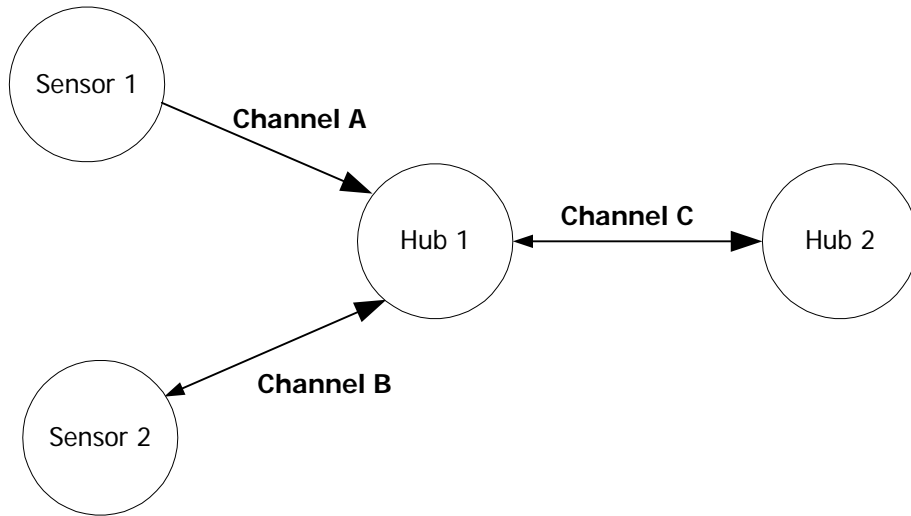
Figure 3-2: A Simple ANT network

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 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-cost 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.
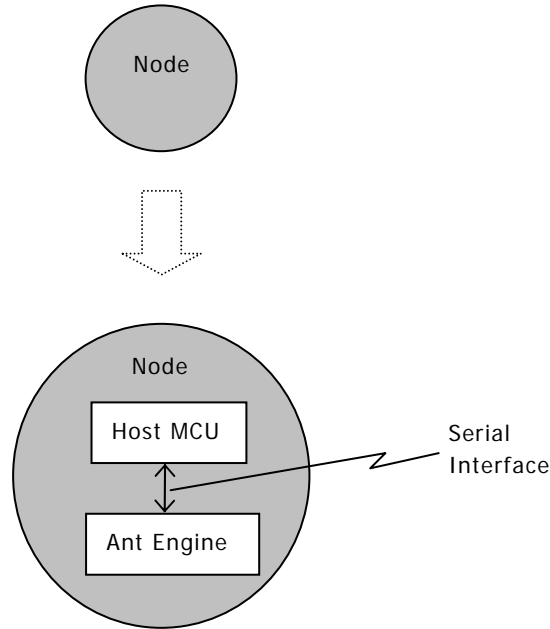
Figure 4-1: Contents of an ANT node

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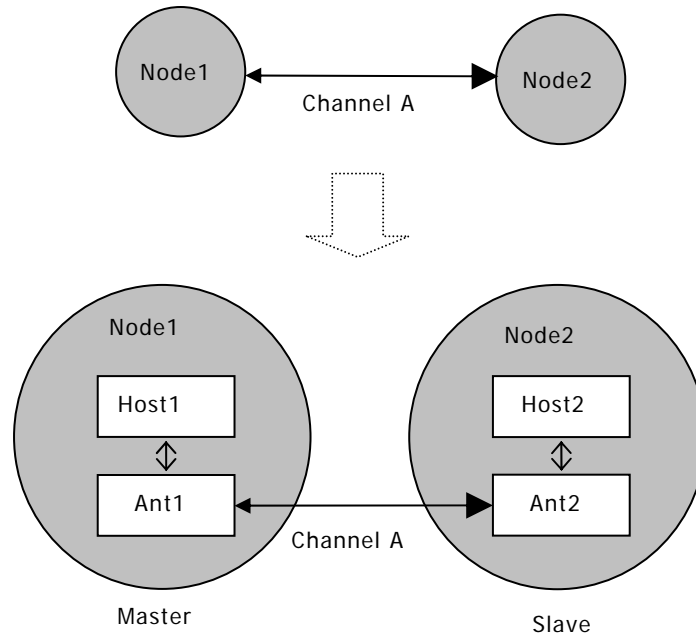
Figure 5-1: Channel communication between two ANT nodes

A channel consists of:

- A master (Node1 above) and

- A slave (Node2 above)

5.1   Channel Configuration

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

- Network

- RF Frequency

- Manufacturer ID

- Device Type

- Channel Type

- Channel Period

- Data types

- Data format

Although the configuration of a specific channel will likely remain constant throughout its connection, a master can maintain multiple channels, each of which may differ in terms of channel configuration parameters.

5.1.1 Network

ANT supports the establishment of numerous unique public 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 purposely shared among multiple vendors with the goal of establishing an 'open' system of interoperable devices, or conversely a private network could be defined to ensure network privacy in order to restrict access only to intended participating devices. 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.

5.1.1.1 Network Number

The Network Number is an 8-bit field with the range of acceptable values being from 0 to the maximum number defined by the ANT implementation. It can be obtained by the host by querying the ANT system using the appropriate request message. Please refer to Section 9 for more details. The default Network Number is 0.

5.1.1.2 Network Key

The Network Key is an 8 byte field which is configurable by the host application. A particular Network Number will have a corresponding Network Key.

The Network Number and the Network Key together provide the ability to deploy 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 networks or initiating your own network, please contact Dynastream at www.thisisant.com.

5.1.2 RF Frequency

ANT technology supports the use of any of the available 125 unique RF operating 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.

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 network frequency being 2524MHz. For example, if a network operating frequency of 2450MHz was desired, the RF frequency field will be set as 50.

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

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

The default RF frequency field value is 66 representing 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.

5.1.3 Channel ID

The most basic descriptor of a channel is its channel ID. In order to establish an ANT channel, the host must specify its channel ID. It's a 4 byte homogenous value that contains 3 fields – Manufacturer ID, Device Type and Device Number. They are detailed below.

5.1.3.1   Manufacturer ID

The manufacturer ID is an 8-bit field used to denote the manufacturer of the device. A unique manufacturer ID space exists for each network. If multiple manufacturers choose to share a network of interoperable devices, any participating slave may search for and pair with devices of differing manufacturers (in effect, ignoring this field). In this scenario, the slave itself must have a manufacturer ID of 0.

5.1.3.2   Device Type

The device type is an 8-bit field used to denote the type of each participating network device. This field is used to differentiate between multiple nodes of network devices so that participants are aware of the various classes of connected nodes and can decode received data accordingly. For example, a heart rate monitor will be assigned a different device type than a bike speed sensor, and its data payload interpreted accordingly.

A receiver node may communicate to multiple device types but its device type must be set to 0.

Please note that the most significant bit of the Device Type is used to control device pairing. More information on device pairing can be found in Section 6.

Specific implementation level information about channel ID usage is provided in the Channel ID functional description located in Section 9.

5.1.3.3   Device Number

The device number is a 16-bit field that is meant to be unique for a given Manufacturer ID and Device Type. Typically, this may be correlated to the serial number of the device or could be a random number generated by the device if the process of setting serial numbers for a particular product is unavailable. This field may also be used as a wild card during device pairing as described in the Channel ID functional description located in Section 9.4.5.3.

5.1.4 Channel Type

Channel type specifies the type of communication that will occur on the channel. It is an 8 bit field with only certain values in the range of 0 to 255 that are acceptable. These are specified in the table below.

Value	Description
0x00	Bidirectional Receive Channel
0x10	Bidirectional Transmit Channel
0x20	Shared Bidirectional Receive Channel
0x30	Shared Bidirectional Transmit Channel
0x40	Receive Channel with option to not clear wild card upon successful match.
0x50	Transmit Only Channel

Please note that the channel type must be specified prior to establishing a channel.

For a bidirectional channel type, the data flow for the forward channel is determined by the mode specified. For example, a node that establishes a bidirectional receive channel type, it will assume the role of the receiver on the forward channel and has the potential to transmit on the reverse channel. Please refer to Sections 5.1.6.1 and 5.1.6.2, below, for more information on the concept of forward and reverse channels.

5.1.5 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 or received on every timeslot at this rate. The channel message rate can range from 0.5Hz to above 200Hz with the upper limit being implementation specific.   The default message rate is 4Hz, which is chosen to provide good performance as described below. It is recommended that the message rate be left at the default, to provide more readily discoverable networks, with good performance characteristics.

The channel period is a 16-bit field with its value determined as shown in 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 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.

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.

5.1.6 Data Types & Data Format

The channel data type determines the type of communication that will occur between the participants of that channel. The data type is specified by the host application using the Message

ID field of the serial message that it provides to the ANT system. Please refer to Section 7 for more details on the serial interface between a host and ANT.

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

5.1.6.1   Forward Channel Communication (Master → Slave)

Data messages are transmitted from the master to slave on every channel timeslot. Three basic data types are supported which are listed below.

1. Broadcast

2. Acknowledged

3. Burst

5.1.6.1.1   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. This form of data is never acknowledged, and so the channel master will be unaware in the case of lost data packets. In the case of a one-way transmission link (transmit-only node communicating to a receiver), broadcast data is the only available data type due to the inability of an acknowledgement.

Broadcast data consumes the least amount of RF bandwidth and system power consumption. It is the preferred choice of communications where occasional data loss is not critical (though 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.

Broadcast data is sent at every time slot of the message period. If no new data has been provided by the host, the previous message is re-transmitted.

5.1.6.1.2   Acknowledged Data

At any time during an established connection, the master can choose to send an acknowledged data packet in place of a broadcast data packet. The slave will respond to the acknowledged data packet with an acknowledgment message back to the master. The master's host controller will be notified of each acknowledged data packet's success or failure. The host may choose to send every data packet as acknowledged data, or mix broadcast and acknowledged data as appropriate to the particular application.

- 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 critical control data where 100% data transmission integrity is required.

There is no automatic re-transmission of unacknowledged data packets.

If the data type isn't specified as Acknowledged or if no new data is provided before the transmit time slot, the message is sent as Broadcast data type.

5.1.6.1.3   Burst Data

Burst data transmission provides a mechanism for the master to send large amounts of data to the slave. A burst transaction begins at the next scheduled timeslot, and consists of a series of continuous acknowledged data messages from master to slave. Any lost messages are automatically retransmitted. There is no limit on the duration of a burst transaction.

A burst transaction takes precedence over all other 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 undesirable behavior.

5.1.6.2   Reverse Channel Data (Slave → Master)

At the end of every channel timeslot, the slave may optionally send data to the master. The same three basic data types as described in Section 5.1.6.1 (Master → Slave) are supported.

1. Broadcast

2. Acknowledged

3. Burst

5.1.6.2.1   Broadcast Data

Broadcast data is sent from the slave to the master at the end of a channel timeslot if expressly requested by the Host MCU of the slave (without a request, no data is sent by default). As is the case for the forward channel, this form of data is never acknowledged, and so the channel slave will be unaware in the case of lost data packets.

Broadcast messages are sent only once on the reverse channel. In other words, there is no broadcast message sent on every time slot of the message period in the reverse channel as opposed to the forward channel.

5.1.6.2.2   Acknowledged Data

At any time during an established connection, the slave can optionally choose to send an acknowledged data packet to the master at the end of a channel timeslot. The master will respond to any acknowledged data packet with an acknowledgment in the form of a valid receipt of the packet. The slave's host controller will be notified of each acknowledged data packet's success or failure. The slave will still always know if the acknowledged data packet has successfully completed.

Similar to the forward channel, there is no automatic re-transmission of unacknowledged data packets.

5.1.6.2.3   Burst Data

Bulk data may be sent from the slave to the master in much the same way as described from master to slave in Section 5.1.6.1.3 (forward channel). A 'reverse' burst transfer will commence at the end of the forward channel timeslot.

The host application software on both the master and slave side should be implemented to expect a common data type (i.e. broadcast vs. acknowledged vs. burst) to be utilized as appropriate for a particular application. In addition, the specific format of the contents of the data payload must of course be previously established by both host controllers such that data can be properly decoded and interpreted.

5.1.7 Example Channel Configuration

Below is an example of a channel configuration for a simple application.

Parameter	Value	Description
Network Type	0	Default Public Network
RF Frequency	66	Default Frequency 2466MHz
Device Number	1	Sample Serial Number
Manufacturer ID	21	Sample Manufacturer ID
Device Type	1	Sample Device Type
Channel Type	0x10	Bidirectional Transmit Channel
Channel Period	16384	2Hz Message Rate
Data Type	0x4E	Broadcast

5.2   Independent Channels

An independent channel has only one master and one slave. It is possible for the master or the 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. All channels have only one master and one slave.

A broadcast network shown in **Figure 3-1** is also formed using independent channels even though the data from one master is received by many slaves. Such a network has a unique master who doesn't purposely initiate communication with multiple slaves on the same channel.

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 support 4 independent channels.

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

5.3   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 two byte Shared Channel Address field and a specific value for the Channel Type both of which are controlled by the host application. The Shared Channel Address field replaces the first two bytes of the data payload provided by the host application to ANT if a channel is defined as being a shared channel. A shared channel can support more than 65000 slave devices using a single independent channel.

In a shared channel network, the node that is required to establish communication with many other nodes must initiate the shared channel as the master. All other nodes that need to access this shared channel must be configured as the slaves. Please note that the channel must be configured as a shared channel at all nodes, master or slave, that may access that channel. The master controls the communication by providing the Shared Channel Address in its forward channel message. A slave will respond on the reverse channel only if its Shared Channel Address matches the one it receives in the forward channel message from the master. In either the master or the slave node, messages are released to the host application only if the Shared Channel Address of the received message matches what the ANT system is configured to respond to at the time or if the Shared Channel Address is 0. Typically, the slave will only be configured once to respond to a particular Shared Channel Address. However, the master will require continuous re-configuration if communication with multiple slaves over the shared channel is desired. The master can send data to all slaves at the same time using the Shared Channel Address of 0.

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 using shared channels.

5.4   Establishing a channel

The following steps must be taken in order to properly establish a communication channel between two ANT nodes.

The prerequisite for establishing a channel is that the master and the slave must have common knowledge of the channel configuration as outlined in Section 5.1.

Procedure:

1. Prior to establishing a channel, the master selects its intended network type, operating frequency, and message rate.

2. The master establishes the channel by transmitting 8-byte payload data packets in a particular timeslot at the established message rate. The ANT channel will be maintained indefinitely at the given timeslot and message rate. The channel master's host controller will optionally provide new data payloads to the ANT engine for continuing transmissions.

3. The slave selects the network type, operating frequency, and message rate of the master from which it'll attempt to receive.

4. The slave establishes a new ANT channel by searching for the intended unique master.

5. Once the master has been located, a connection is established, and the slave receives data indefinitely from the master at the given timeslot and message rate.

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 in future communication sessions. This relationship can be permanent, semi-permanent or transitory.

A pairing operation consists of a slave device acquiring the unique ID of the master device. If permanent pairing is desired, the slave node should store the master's ID to be used to open a channel with this ID in all subsequent communication sessions. In a semi-permanent situation, the slave may wish to occasionally purge any stored ID and pair with a new master. In a transitory situation, the slave may pair with a master on a temporary basis only.

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.

Establishing a channel involves the broadcasting of a unique ID by the master, and a search and acquisition of this ID by a slave. In the case where a slave does not have knowledge of a specific master ID, a paring mechanism is available. The slave can search for a master with a wild card ID, and 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.

6.1   Pairing Example

We present an example of a pairing operation by proposing a network of three remote temperature sensors (masters) and one base unit (slave), as shown in the following diagram.
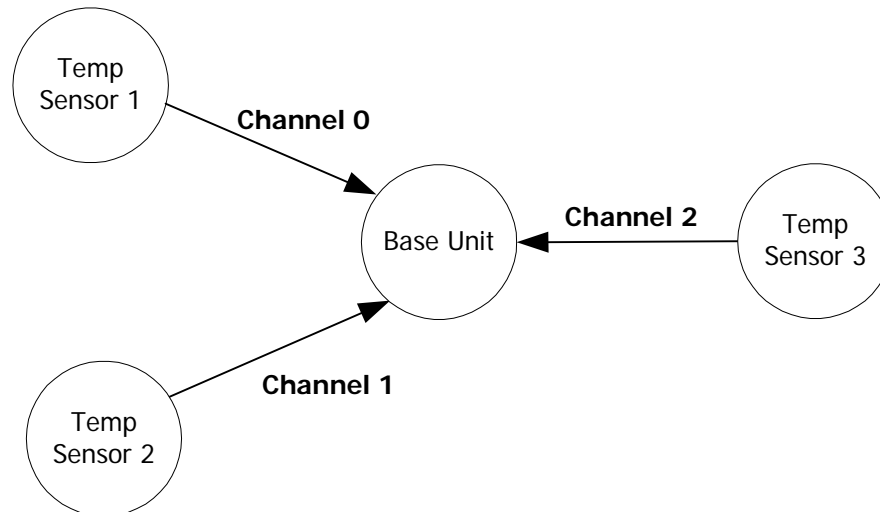
Figure 6-1: Example ANT Network for use in Device Pairing

We'll assume that 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)

3. Open TX Channel

4. Begin transmitting data on its' 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 (Manuf ID = Specific or Wild card, Device Type = Temperature sensor with Pairing Bit Set, Device Number = Wild Card)

3. Open RX Channel

The base unit finds a temperature sensor ID as reported by the slave ANT engine to the host controller, which is stored by the host controller for future channel establishment. This procedure is repeated for all three temperature sensors.

The temperature sensor can then 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 optionally only required to be performed once for the lifetime of an ANT system if a permanent relationship between two specific devices is desired. In such a case, pairing may be performed during product manufacturing (factory environment) to remove the burden of requiring the customer to do so.

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.

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	CHECKSUM
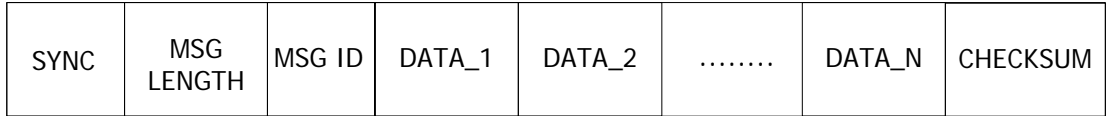
Figure 7-1: ANT Serial Message Structure

As shown above, each message begins with a SYNC byte and ends with the CHECKSUM. The bytes are sent LSBit 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 (MSB:LSB)
1	MSG LENGTH	1 Byte	Number of data bytes in the message. 1 < N < 9
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 of 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.

7.2 Host MCU Serial Interface – Physical Layer

The ANT serial interface between host controller and ANT engine can be implemented over either a synchronous or asynchronous connection, which can be selected by the product designer as preferred for a given implementation. The precise details of the physical and electrical interface of each ANT product can be found in each respective ANT product datasheet.

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.

8 Example ANT Network Implementation

In order to clearly present the features of the ANT engine a sample network implementation is shown below.

Consider a simple 4 node network as shown in Figure 8-1 below.
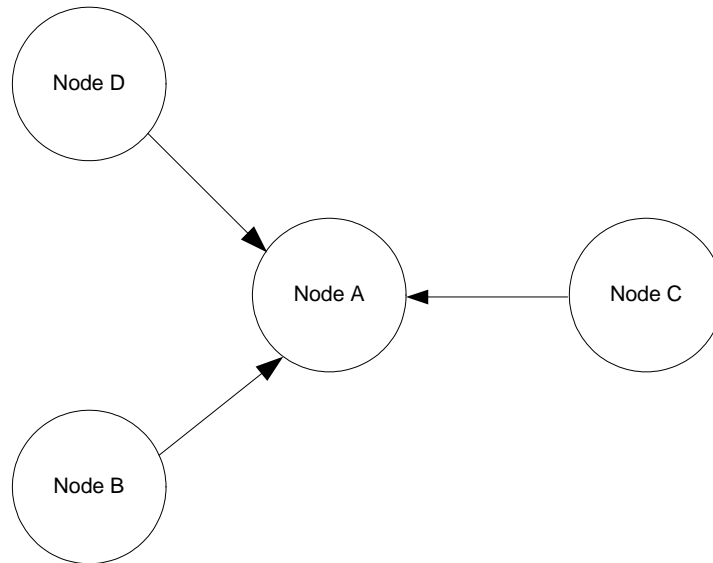
Figure 8-1: Example ANT Network for implementation

The example network shown above describes an application where information from multiple nodes (Nodes B, C and D) is to be received and possibly analyzed by a single central node (Node A). The arrows indicate the primary flow of information between the corresponding nodes.

The following can be assumed about this example network.

- Node B is to use the broadcast data type

- Node D is to use 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. are known between all nodes

- Device pairing has already been performed between the Masters and their corresponding slaves

In the following, two methods of utilizing ANT to deploy the above example network are described.

8.1 Implementation using Independent Channels

Using independent channels will be the simplest method by which the aforementioned network can be implemented. Given the above assumptions, three independent channels will be required with configuration for each as 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 Type	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	1	Serial Number of Node B
	Manufacturer ID	1	Sample Manufacturer ID
	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 Type	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	1	Serial Number of Node B
	Manufacturer ID	1	Sample Manufacturer ID
	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 Type	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	10	Serial Number of Node C
	Manufacturer ID	1	Sample Manufacturer ID
	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 Type	0	Default Public Network
	RF Frequency	72	Frequency 2472MHz
	Device Number	10	Serial Number of Node C
	Manufacturer ID	1	Sample Manufacturer ID
	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 A will be the master:

Node	Parameter	Value	Description
D	Network Type	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	2	Serial Number of Node D
	Manufacturer ID	1	Sample Manufacturer ID
	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 Type	0	Default Public Network
	RF Frequency	66	Default Frequency 2466MHz
	Device Number	2	Serial Number of Node D
	Manufacturer ID	1	Sample Manufacturer ID
	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

Listed below are the sequence of events and message transactions between the host and ANT for each participating node as the above channels are established and the network is formed. Please refer to Section 5.4 for more information on the procedure for establishing a channel.

8.1.1 Channel between Node B and Node A

The channel between Node B and Node A is established as follows.

Master: Node B

1. The host issues the ANT_AssignChannel() message with the configuration fields filled as mentioned in the table above. The channel number is assigned at the discretion of the host.

2. The host issues the ANT_SetChannelID() message with the configuration fields filled as mentioned in the table above. The channel number is the same as what was used in step 1.

3. The host opens the channel using the ANT_OpenChannel() message.

4. The 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 transmit the same data if no new data is made available by the host. However, appropriate safeguards to account for such repeated messages should be in place on the slave.

Slave: Node A

1. The host issues the ANT_AssignChannel() message with the configuration fields filled as mentioned in the table above. The channel number is assigned at the discretion of the host.

2. The host issues the ANT_SetChannelID() message with the configuration fields filled as mentioned in the table above. The channel number is the same as what was used in the previous step.

3. The host opens the channel using the ANT_OpenChannel() message.

4. The host will now be informed using the ChannelEventFunc(Chan, EV) type message from ANT whenever the message from Node B is received. Based on the channel configuration settings, this will happen at 1Hz.

8.1.2 Channel between Node C and Node A

The channel between Node C and Node A is established as follows.

Master: Node C

1. The host issues the ANT_AssignChannel() message with the configuration fields filled as mentioned in the table above. The channel number is assigned at the discretion of the host.

2. The host issues the ANT_SetChannelID() message with the configuration fields filled as mentioned in the table above. The channel number is the same as what was used in step 1.

3. The host opens the channel using the ANT_OpenChannel() message.

4. The host provides ANT with application data that is to be sent through the channel as an acknowledged message using the ANT_SendAcknowledgedData() message. ANT will then communicate the success or failure of the transaction back to the host using a message code embedded inside a channel event message. If ANT didn't receive an acknowledge back from the slave, it will not automatically resend the message. The host must process the failure message code and handle the retries itself. Please refer to the Appendix for a list of the various message codes.

Slave: Node A

1. The host issues the ANT_AssignChannel() message with the configuration fields filled as mentioned in the table above. The channel number is assigned at the discretion of the host.

2. The host issues the ANT_SetChannelID() message with the configuration fields filled as mentioned in the table above. The channel number is the same as what was used in the previous step.

3. The host opens the channel using the ANT_OpenChannel() message.

4. The host will now be informed using the ChannelEventFunc(Chan, EV) type data message whenever an acknowledged message is received from Node C. The acknowledgement of the received message is handled within ANT; hence the host only needs to process or use the received data. The messages will be received at the rate specified by the channel configuration settings.

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 C will open a third channel to communicate with Node B in the same way as for Node A.

The above ANT network will continue to function as it was deployed unless an application layer event dictates otherwise.

8.2   Implementation using Shared Channels

Another method by which the network shown in Figure 8-1 can be implemented is with the use of a single shared channel instead of 3 independent channels. The trade-off is the increased computational burden on the host of the network hub (Node A) and the reduction in the amount of maximum useful data per packet from 8 bytes to 6 bytes.

As mentioned in the description of Shared Channels earlier, the main receiver will become the master of the shared channel with the transmitters its slaves. Each slave will have a unique two byte shared channel address which will be known only to itself and the master. Moreover, all nodes on the shared channel network will have to be transceivers as opposed to the independent channel implementation where nodes B and D could be transmit only. The updated network diagram for this setup is shown below.
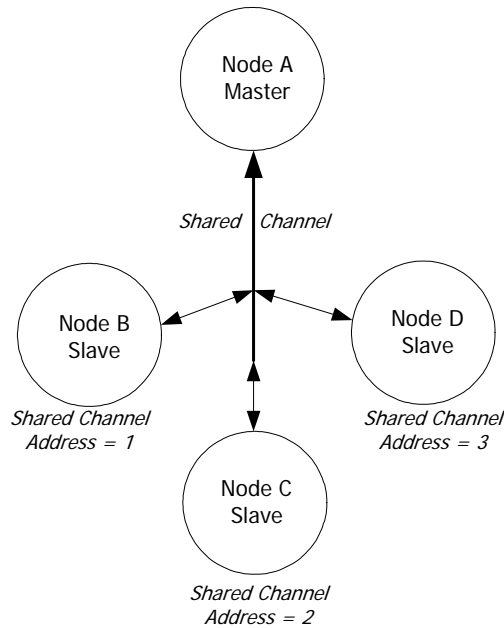

Figure 8-2: Shared channel implementation of example network

The channel configuration for each node is shown in the table below.

PRELIMINARY

Slave Node B:

Parameter	Value	Description
Network Type	0	Default Public Network
RF Frequency	66	Default Frequency 2466MHz
Device Number	3	Serial Number of Node A
Manufacturer ID	1	Manufacturer ID on Node A
Device Type	3	Device Type of Node A
Channel Type	0x20	Shared Receive Channel
Channel Period	2370	~12Hz Message Rate
Data Type	0x4E	Broadcast

Slave Node C:

Parameter	Value	Description
Network Type	0	Default Public Network
RF Frequency	66	Default Frequency 2466MHz
Device Number	3	Serial Number of Node A
Manufacturer ID	1	Manufacturer ID of Node A
Device Type	3	Device Type of Node A
Channel Type	0x20	Shared Receive Channel
Channel Period	2370	~12Hz Message Rate
Data Type	0x4F	Acknowledged

Slave Node D:

Parameter	Value	Description
Network Type	0	Default Public Network
RF Frequency	66	Default Frequency 2466MHz
Device Number	3	Serial Number of Node A
Manufacturer ID	1	Manufacturer ID of Node A
Device Type	3	Device Type of Node A
Channel Type	0x20	Shared Receive Channel
Channel Period	2370	~12Hz Message Rate
Data Type	0x4E	Broadcast

Master Node A:

Parameter	Value	Description
Network Type	0	Default Public Network
RF Frequency	66	Default Frequency 2466MHz
Device Number	3	Serial Number of Node A

Manufacturer ID	1	Manufacturer ID of Node A
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, Manufacturer ID and Device Type are controlled by the shared channel master (Node A). All slaves that want to use this shared channel must adhere to these parameters.

- The application level channel periods are the same for Node B (4Hz), Node C (4Hz) and Node D (4Hz) as they were in the implementation using independent channels. However, the master (Node A) will need the shared channel to operate at 12Hz (the sum of the message rates of all slave nodes) at the very least to suffice the requirements for each slave. Since there is only one channel, the shared channel, all slaves will have to abide by the rules of the channel initiated by the master. Hence, they will have to establish channels with the same message rate as the master. However, the host of each of the slave nodes can still communicate with ANT at only 4Hz.

The following steps need to be taken for implementing the aforementioned 4 node ANT network using a single shared channel.

Master Node A:

1. The host issues the ANT_AssignChannel() message with the configuration fields filled as shown in the table above. The channel number is assigned at the discretion of the host.

2. The host issues the ANT_SetChannelID() message with the configuration fields filled as shown in the table above. The channel number is the same as what was used in step 1.

3. The host assigns the channel's message rate using the ANT_SetChannelPeriod() message and the value listed in the above table.

4. The host opens the channel using the ANT_OpenChannel() message.

5. The host provides ANT with an ANT_SendBroadcastData() message with the first two data bytes representing the shared channel address of the slave to which it wishes the next RF message be addressed.

6. ANT will inform the host each time a message is received on the reverse channel of the RF message that was sent to the slave with the corresponding shared channel address. For this particular network, each slave would send a message back to the master Node A each time it's shared channel address appears.

Slave Node B:

1. The host issues the ANT_AssignChannel() message with the configuration fields filled as shown in the table above. The channel number is assigned at the discretion of the host.

2. The host issues the ANT_SetChannelID() message with the configuration fields filled as shown in the table above. The channel number is the same as what was used in step 1.

3. The host sets the channel period using the ANT_SetChannelPeriod() to the same message rate as the master Node A.

4. The host opens the channel using the ANT_OpenChannel() message.

5. The host sends a single Send_BroadcastData() message with the first two bytes as the shared channel address of Slave Node B to configure ANT to listen to the messages that are addressed to Slave Node B.

6. The host will now be informed each time ANT receives a message from the master on this channel that has the shared channel address of Node B. For this application, the host would provide data to ANT to be sent back to the master on the reverse channel using the ANT_SendBroadcastData() message at a 4Hz rate. ANT will send the data on the reverse channel whenever it receives the properly addressed message on the forward channel.

Slave Node C:

1. The host issues the ANT_AssignChannel() message with the configuration fields filled as shown in the table above. The channel number is assigned at the discretion of the host.

2. The host issues the ANT_SetChannelID() message with the configuration fields filled as shown in the table above. The channel number is the same as what was used in step 1.

3. The host sets the channel period using the ANT_SetChannelPeriod() to the same message rate as the master Node A.

4. The host opens the channel using the ANT_OpenChannel() message.

5. The host sends a single Send_BroadcastData() message with the first two bytes as the shared channel address of Slave Node C to configure ANT to listen to the messages that are addressed to Slave Node C.

6. The host will now be informed each time ANT receives a message from the master on this channel that has the shared channel address of Node C. For this application, the host would provide data to ANT to be sent back to the master on the reverse channel using the ANT_SendAcknowledgedData() message at a 4Hz rate. ANT will send the data on the reverse channel whenever it receives the properly addressed message on the forward channel. A channel even message from ANT will inform the host of whether an acknowledgement was received from the master or that the data was sent back successfully.

Slave Node D:

The procedure is exactly the same as that for slave Node B with the only difference being that the ANT will be configured to inform the host when it receives a message with the shared channel address of Node D.

The above two implementations of the example network are to be used as means for gaining familiarity with network design and deployment using ANT. This particular example network could of course be implemented in many 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, manufacturer ID, message rate and 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 the activity of the ANT system.

9.1.3   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 of these messages.

9.1.4   Channel Event/Response Messages

The channel event/response messages comprise of notifications and data that is 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.

9.1.5   Requested Response Messages

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

9.1.6   Test Mode

ANT also accepts special test mode messages which allow the product developer or tester to verify proper operation of the RF hardware but placing ANT in a RF 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.

Page 31 of 51

PRELIMINARY

9.3 ANT Message Summary

Class	Type	ANT PC Interface Function	Response	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()	Yes	Host	1	0x41	Channel Number								
	Assign Channel	ANT_AssignChannel()	Yes	Host	3	0x42	Channel Number	Channel Type	Network Number						
	Channel ID	ANT_SetChannelId()	Yes	Host	5	0x51	Channel Number	Device number		Device Type ID	Man ID				
	Channel Period	ANT_SetChannelPeriod()	Yes	Host	3	0x43	Channel Number	Messaging Period							
	Search Timeout	ANT_SetChannelSearchTimeout()	Yes	Host	2	0x44	Channel Number	Search Timeout							
	Channel RF Frequency	ANT_SetChannelRFFreq()	Yes	Host	2	0x45	Channel Number	RF Frequency							
	Set Network	ANT_SetNetworkKey()	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()	Yes	Host	2	0x47	0	TX Power							
Control Messages	SystemReset	ANT_ResetSystem()	No	Host	1	0x4A	0								
	Open Channel	ANT_OpenChannel()	Yes	Host	1	0x4B	Channel Number								
	Close Channel	ANT_CloseChannel()	Yes	Host	1	0x4C	Channel Number								
	Request Message	ANT_RequestMessage()	Yes	Host	2	0x4D	Channel Number	Message ID							
Data Messages	Broadcast Data	ANT_SendBroadcastData() ->ChannelEventFunc(Chan, EV)	No	ANT/ Host	9	0x4E	Channel Number	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
	Acknowledge Data	ANT_SendAcknowledgedData() ->ChannelEventFunc(Chan, EV)	No	ANT/ Host	9	0x4F	Channel Number	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
	Burst Transfer Data	ANT_SendBurstTransferPacket() ->ChannelEventFunc(Chan, EV)	No	ANT/ Host	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);	-	ANT	3	0x40	Channel Number	Message ID	Message Code						
Requested Response Messages	Channel Status	->ResponseFunc(Chan,0x52)	-	ANT	2	0x52	Channel Number	Channel Status							
	Channel ID	->ResponseFunc(Chan,0x51)	-	ANT	5	0x51	Channel Number	Device number		Device Type ID	Man ID				
	ANT Version	->ResponseFunc(-, 0x3D)	-	ANT	9	0x3D	Ver0	Ver1	Ver2	Ver3	Ver4	Ver5	Ver6	Ver7	Ver8
	Capabilities	->ResponseFunc(-, 0x54)	-	ANT	4	0x54	Max Channels	Max Networks	Standard Options	Advanced Options					
Test Mode	CW Init	ANT_InitCWTestMode()	Yes	Host	1	0x53	0								
	CW Test	ANT_SetCWTestMode()	Yes	Host	3	0x48	0	TX Power	RF Freq						

P +1 403.932.9292   F +1 403.932.6521

thisisant.com

ANT   Dynastream Innovations Inc. 228 River Avenue, Cochrane, Alberta, Canada

9.4   ANT Message Details

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

9.4.1 ANT Constants

The following 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.

9.4.2 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.

9.4.2.1   Unassign Channel (0x41)
        void ANT_UnAssignChannel(UCHAR ucChannel);

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

```
// Example usage
ANT_AssignChanel(0, 0x00, 0);
..
ANT_UnAssignChannel(0);
```

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

9.4.2.2   Assign Channel (0x42)
void ANT_AssignChannel(UCHAR ucChannel, UCHAR ucChannelType, UCHAR ucNetworkNumber);

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	0x00 - Receive Channel 0x10 - Transmit Channel 0x50 – Transmit Only Channel

			0x40 – Receive Channel with Do not clear wild card on match Option Shared Channels 0x20 – Shared Receive Channel 0x30 – Shared 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.

```
// Example Usage
ANT_AssignChannel(0, 0x00, 0); // receive channel on network number 0
```

This message is sent to the module to assign a channel. Channel assignment reserves a channel number and assigns the type and network number to the channel. This Assign Channel command should be issued before any of the other channel configuration messages, and before the channel is opened. Assigning a channel sets all of the other configuration parameters to their defaults.

9.4.2.3   Channel ID (0x51)

void   ANT_SetChannelID(UCHAR   ucChannel,   USHORT   usDeviceNum,   UCHAR   ucDeviceType,   UCHAR ucManufacturer);

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 receiver use 0 to match any device number.
Pairing Request	UCHAR (1bit)	7	0..1	Pairing Request. Set this bit on transmitter to request pairing Set this bit on receiver to find a pairing transmitter.
Device Type ID	UCHAR (7bits)	0-6	0..127	The device type. For a receiver use 0 to match any device type.
Manufacturer ID	UCHAR	-	0..255	The manufacturer ID. For a receiver use 0 to match any manufacturer.

```
// Example Usage
// Tx channel
ANT_AssignChannel(0, 0x10, 0);
  // wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 1234, 120, 123);
/****************************************************/
// Rx channel
ANT_AssignChannel(0, 0x00, 0);
  // wait for RESPONSE_NO_ERROR
ANT_SetChannelId(0, 0, 120, 123); // 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, 123); // device number is wild-card, device type 120 with pairing bit ON
```

This message configures the channel ID to be used by a specific channel.

The Channel ID is intended to be unique (or nearly so) for every device link in a network. The ID is owned by the transmitter. The transmitter sets its ID, and it is transmitted along with its messages. The receiver sets the channel ID to match the transmitter 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 Manufacturer). When a match is found using a wildcard search, the Request Message command with a Message ID parameter of Channel ID, will cause ANT to return a Channel ID message containing the ID of the matched device.

If the Device Number is set to 0 on the receiver, the receiver will search for any transmitters that match its Device Type and Manufacturer. It must also match the state of the Pair Request bit. 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 transmitter and receiver will both set the pairing bit when they wish to pair. If the designer intends for any receiver of that type to pair to any transmitter of that type at any time on a search, then the pairing bit should always be set to 0.

When the Device Number is fully known the Pairing Bit is ignored. (ie if you know the exact device you are looking for, then pairing is irrelevant).

Please note – that except for the free default network - Manufacturer ID and Type IDs are assigned and regulated to maintain network integrity, and interoperability. Please visit www.thisisant.com for more details on available standard network types or on how to obtain your own network type identifier.

9.4.2.4   Channel Messaging Period (0x43)

void 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 message period
```

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

Messaging Period = Message Period Time (s) * 32768.

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

Note: The minimum acceptable message 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.

The message period is of critical importance to define in a manner consistent with the needs of the application. Some issues to consider are listed next:

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 receiver does not wish to receive data as fast as it is being transmitted, it may receive data at an integer multiple slower data rate (do not use non-integer multiples). For example, if the transmitter is transmitting the data at 4Hz (8192), the receiver may prefer to receive data at 1Hz (32768). The receiver 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 receiver. Of course, the required data refresh rate on the receiver needs to be considered if data messages are to be skipped.

9.4.2.5 Channel Search Timeout (0x44)
          void 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	12 (30seconds)	The search timeout to be used with by this channel for receive searching. A value of 0 will result is no timeout. Each count in this parameter is equivalent to 2.5 seconds. I.E. 240 = 600 seconds = 10 minutes

```
// 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 the receiver will search for a channel before timing out.

9.4.2.6 Channel RF Frequency (0x45)
          void 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

PRELIMINARY

```
// 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.

This is an advanced feature and the default RF frequency should be used, unless great care is taken in choosing an alternate. The selection of this channel may affect the ability to certify the product in certain regions in the world.

9.4.2.7   Set Network Key(0x46)
void 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.

To use the default public network this command is not required. The default public network key is already assigned by default to Network Number 0.

Please note – that except for the free default network – Network Keys, Manufacturer ID, and Type IDs are assigned and regulated to maintain network integrity, and interoperability. Please visit www.thisisant.com for more details on available standard network types or on how to obtain your own network type identifier.

9.4.2.8   Transmit Power (0x47)
void 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	2 (-5dB)	0 = TX Power -20 dBm 1 = TX Power -10 dBm 2 = TX Power -5 dBm 3 = TX Power 0 dBm

```
// Example Usage
ANT_SetTransmitPower(3); // set the RF output power to 0 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.

9.4.3 Control Messages

9.4.3.1 Reset System(0x4A)
```
void 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 will terminate all channels. All information previously configured in the system can no longer be considered valid.

9.4.3.2 Open Channel (0x4B)
```
void 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. Execution of this command will cause the channel to commence operation, and either data messages or events will begin to be issued in association with this channel.

9.4.3.3 Close Channel (0x4C)
```
void 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. 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.

9.4.3.4   Request Message (0x4D)

void 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	0x3D, 0x3F, 0x51, 0x52	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.

There are currently four valid Messages which can be requested: Channel Status (0x52), Channel ID (0x51), ANT Version (0x3D), and Capabilities (0x3F). Requesting one of these messages will cause ANT to send the appropriate response message. Please see these messages for specific details.

9.4.4 Data Messages

There are three methods of sending or receiving data on a channel. These methods are described below.

9.4.4.1   Broadcast Data (0x4E)

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

or

ChannelEventFunc (Channel, EVENT_RX_BROADCAST) // Receive

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

```
// 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_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 unacknowledged, and is not guaranteed to be received by the receiver.

Transmitter

An ANT channel set up for transmit will send a message to the receiver at the programmed channel period. After every transmission it will generate an EVENT_TX message to prompt the transmit MCU that ANT is ready for the next data packet. The Broadcast Data message can be used to send ANT the next Data packet for transmission. Failure to send the Broadcast Data message before the next slot will cause the old data to be transmitted. It is up to the transmit MCU on every message to send new data.

Receiver

The corresponding channel receiver will receive the data at its programmed channel period and generate a Broadcast Data message to the receive MCU. If the receiver 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, followed by sending a special library only event EVENT_RX_BROADCAST to let you know that a valid broadcast message has been received.

9.4.4.2   Acknowledged Data (0x4F)
void ANT_SendAcknowledgedData(UCHAR ucChannel, UCHAR* pucBroadcastData); // Transmit

or

ChannelEventFunc(Channel, EVENT_RX_ACKNOWLEDGED) // Receive

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

```
// 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_ACKNOWLEDGED:  // PC applications only; use MsgID 0x4F in embedded
      Switch (ucChannel)
      {
        Case Channel_0:
          // process received data which is in channel event buffer
        Break;
        ..
      }
    Break;
  }
}
```

If it is desired to ensure the successful transmission of data, the Acknowledged Data message can be used in place of the Broadcast Data message. This can be done for every message, or only when acknowledgement of the transmission is required. Acknowledged data is transmitted in the same transmission time slot as Broadcast Data would be, although it extends the length of the timeslot to accommodate the acknowledgement. Acknowledged Data transmissions can not be originated from channels configured for transmit only.

Transmitter

PRELIMINARY

Behaves exactly as mentioned above for the Broadcast Data message, with the exception that the EVENT_TRANSFER_TX_COMPLETED message is generated for a successful transmission. If the transmission fails, an EVENT_TRANSFER_TX_FAILED message is generated. Either EVENT prompts the transmit MCU that ANT is ready for the next data packet.

Receiver

Reception of Acknowledged Data form the transmitter will cause an Acknowledged Data Message to be sent to the receiver MCU. If the message reception fails, either an EVENT_RX_FAIL will occur.

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

9.4.4.3 Burst Data (0x50)

```
Void ANT_SendBurstTransfer(UCHAR ucChannel, UCHAR* pucData, USHORT usNumDataPackets);  // Transmit
```

or

```
ChannelEventFunc (Channel, EVENT_RX_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.
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_SendBurstData(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_BURST_PACKET:  // PC applications only; use MsgID 0x50 in embedded
      Switch (ucChannel)
      {
        Case Channel_0:
           // process received data which is in channel event buffer one packet at a time validating the
           // sequence
        Break;
        ..
      }
     Break;
  }
}
```

Burst data transmission can be used to send larger amounts of data, by sending messages continuously at the fastest rate possible. Each message in a Burst Transfer is acknowledged, and all lost message are resent to guarantee reception of the entire data. 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.

Transmitter

If the transmitter wishes to send multiple packets of data in fast succession, the Burst Data packet can be sent in place of a Broadcast Data or Acknowledged Data message. The host MCU is 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. If an EVENT_TRANSFER_TX_FAILED is received at any time during this sequence, the transfer has failed and the burst transfer has been terminated. If the application wishes to retry, it must restart the burst transfer sequence.

Receiver

Reception of Burst Data form the transmitter will cause Burst Data Messages to be sent to the receiver MCU. If burst message reception exceeds the maximum number of retries an EVENT_TRANSFER_RX_FAIL will occur.

Sequence Numbers

The upper 3 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]

9.4.5 Channel Response / Event Messages

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

9.4.5.1   Channel Response / Event (0x40)

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

or

ResponseFunc (Channel, MessageCode) // 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. Set to 1 if being sent 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)

* Additional message codes may be generated by certain ANT products. Please refer to the datasheet for special notes in this regard.

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 would happen when a receiver is tracking a transmitter 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 transmitter.
EVENT_TRANSFER_RX_FAILED	4	A receive transfer has failed. This occurs when a Burst

PRELIMINARY

Name	Value	Description
		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 has failed to complete successfully
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. This event is the actual indication of the closure of the channel. So, the Host must use this event message instead of the RESPONSE_NO_ERROR message to let a channel state machine continue.
EVENT_RX_BROADCAST	10	ANT Library special event (Not in serial interface). This event is sent to denote a valid broadcast data message has been received by the ANT library, and the data is waiting in the appropriate channel receive buffer.
EVENT_RX_ACKNOWLEDGED	11	ANT Library special event (Not in serial interface). This event is sent to denote that a valid acknowledged data message has been received by the ANT library, and the data is waiting in the appropriate channel receive buffer.
EVENT_RX_BURST_PACKET	12	ANT Library special event (Not in serial interface). It indicates the successful reception of a burst packet in a Burst Transfer sequence.
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	Attempt to transmit data on an unopened channel
CHANNEL_ID_NOT_SET	24	Returned on attempt to open a channel before setting a valid ID
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
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.

```
// Example Usage
BOOL ANT_ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
  Switch (ucEvent)
  {
    Case EVENT_RX_BROADCAST:
```

Name	Value	Description

```
    Switch (ucChannel)
    {
     Case Channel_0:
        // process data which is in aucChannelEventBuffer
     Break;
     ..
     Case Channel_N:
        // process data which is in aucChannelEventBuffer
     Break;
     }
   Break;
   Case EVENT_RX_FAIL:
    Switch (ucChannel)
    {
     Case Channel_0:
        // data packet was lost
     Break;
     ..
     Case Channel_N:
        // data packet was lost
     Break;
     }
   Break;
   Case Default:
        // catch unexpected message codes
   Break;
  }
}
```

9.4.6 Requested Response Messages

The following messages are returned in response to Request Message (message 0x25) being sent to ANT, with the appropriate parameter for the desired message. The ANT PC library will call the Host application's ANT response function with message ID as indicated below for each message.

The message ID codes are defined in antmessage.h.

9.4.6.1   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)	0	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;
  }
}
```

9.4.6.2   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*
Manufacturer ID	UCHAR	0..255	The manufacturer ID*

This message returns the channel ID of the specified channel. This message is useful when trying to pair devices. When a receiver is attempting to pair to a transmitter, it will typically set one or more of the Device Number, Device Type ID, or Manufacturer ID fields with a wild card. When the receiver finds a device that matches the search – by successfully receiving data, the Request Message can be used to return the found Channel ID. This ID can then be saved for future use in opening a channel which will search for this specific device. See pairing under the Usage section for more details.

* Please note that except for the free default network - Manufacturer ID and Device Type IDs are assigned and regulated to maintain network integrity, and interoperability. Please visit www.thisisant.com for more details on available standard network types or on how to obtain your own network type identifier.

9.4.6.3   ANT Version (0x3D)
ResponseFunc (-, 0x3D)

The version message returns a 9 byte null terminated version string, corresponding to the ANT host interface version. Please note that this message is not supported on all ANT products, especially the nRF24AP1.

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

9.4.6.4 Capabilities (0x54)
ResponseFunc (-, 0x54)

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

Parameters	Type	Range	Description
Max Channels	UCHAR	0..MAX_CHAN-1	Returns the Number of channels available on ANT
Max Networks	UCHAR	0..MAX_NET-1	
Standard Options	UCHAR	0..255	The Standard Options bitfield is encoded as follows: #define CAPABILITIES_NO_RECEIVE_CHANNELS 0x01 #define CAPABILITIES_NO_TRANSMIT_CHANNELS 0x02 #define CAPABILITIES_NO_RECEIVE_MESSAGES 0x04 #define CAPABILITIES_NO_TRANSMIT_MESSAGES 0x08 #define CAPABILITEIS_NO_ACKD_MESSAGES 0x10 #define CAPABILITIES_NO_BURST_MESSAGES 0x20 Other bits are reserved
Advanced Options	UCHAR	0..255	The Advanced Options bitfield is encoded as follows: #define CAPABILLITES_NETWORK_ENABLED 0x02 Other bits are reserved

9.4.7 Test Mode

9.4.7.1 Init CW Test Mode (0x53)
void ANT_InitCWTestMode(void);

Parameters	Type	Range	Description
Filler	UCHAR	0	

This function is required to be called before the CW Test Mode message below, to put the module in 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.

9.4.7.2 CW Test Mode (0x48)
void 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 the channel RF frequency provided.

This command is intended to test your implementation for RF regulatory requirements. It will set the transmitter to transmit 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.

9.4.8 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 since the messages will be exchanged directly through a serial interface.

9.4.8.1   ANT PC Library Usage Notes

The following notes apply when using the ANT PC Library. The files for this library are located on the ANT Development Kit CD.

- ANT_DLL.dll, SiUSB.dll and SiUSBXp.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.

- ANT_Interface.h must be included where calls to the ANT PC library are made.

9.4.8.2   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 module, otherwise FALSE.

9.4.8.3   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.

9.4.8.4   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 will be 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.

9.4.8.5   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 will be written to. 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 will be 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 other ANT functions are called which use this channel.

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.

9.5   ANT Library Access

As part of the development kit, developers are able to utilize the ANT library functions described in section 9.4. This access is provided primarily through the ANT_Interface.dll, as well as several header files. These files can all be found in the DLL folder on the Development Kit CD. The required header files for general use are antmessage.h, antdefines.h, and types.h. The ANT header files (antmessage.h and antdefines.h) provide the define statements that are useful in decoding the messages returned by the library functions. The .dll itself can be accessed using both static and dynamic linking. For static linking, the ANT_Interface.lib and ANT_Interface.h files are required to access the functions. This method is only available for applications developed using Borland Builder. Dynamic linking is available for applications developed in Borland and Visual C++. The function names for dynamic linking are the same as described in section 12.5 except each name is preceded by an underscore, thus the lookup function should use the underscore version to properly access the function. Some sample connecting code is shown below for ANT_Close(). When defining the function protocol (typedef statement), __stdcall must not be used.

Example:

typedef void (*ANT_vFn)();

ANT_vFn ANT_Close = NULL;

void DLLInit()

{

HINSTANCE hDLL = NULL;

hDLL = LoadLibrary("ANT_Interface.dll");

if(hDLL == NULL)

                return;

        else {

                ANT_Close = (ANT_vFn)GetProcAddress(hDLL, "_ANT_Close");

PRELIMINARY

```
        if(ANT_Close == NULL){

                FreeLibrary(hDLL);

                return;

        }

    }
}
```