



ANT Message Protocol and Usage

D00000652 Rev 2.9
Dynastream Innovations Inc.
July 2, 2007

P +1 403.932.4620 F +1 403.932.6521

Copyright Information and Usage Notice

This information disclosed herein is the exclusive property of Dynastream Innovations Inc. No part of this publication may be reproduced or transmitted in any form or by any means including electronic storage, reproduction, execution or transmission without the prior written consent of Dynastream Innovations Inc. The recipient of this document by its retention and use agrees to respect the copyright of the information contained herein.

The information contained in this document is subject to change without notice and should not be construed as a commitment by Dynastream Innovations Inc. unless such commitment is expressly given in a covering document.

The Dynastream Innovations Inc. ANT Products described by the information in this document are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Dynastream product could create a situation where personal injury or death may occur. If you use the Products for such unintended and unauthorized applications, you do so at your own risk and you shall indemnify and hold Dynastream and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Dynastream was negligent regarding the design or manufacture of the Product.

©2007 Dynastream Innovations Inc. All Rights Reserved.



Revision History

| Revision | Effective Date | Description |
|----------|--------------------|--|
| 1.0 | April 9, 2005 | Preliminary Version Release |
| 1.31 | April 29, 2005 | External Release |
| 2.0 | December 6, 2005 | Description of shared channel broadcast address, new event codes, addition of various examples and other minor updates |
| 2.1 | September 18, 2006 | Deleted broadcast acknowledged and burst channel events from the table. |
| 2.2 | November 14, 2006 | Added new messages for release |
| 2.3 | March 30, 2007 | Added SensRcore™ message support |
| 2.4 | April 24, 2007 | Added Scanning Channel Details |
| 2.5 | May 28, 2007 | Added Extended Data Message Description |
| 2.6 | June 7, 2007 | Updated EVENT table and added table describing which events are available on which products. |
| 2.7 | June 27, 2007 | Minor fixes |
| 2.8 | July 2, 2007 | Document format change |
| 2.9 | July 6, 2007 | Document format change |



Table of Contents

| | | |
|-----------|---|-----------|
| 1 | Introduction | 5 |
| 2 | The ANT Product Family | 6 |
| 3 | Network topologies | 7 |
| 4 | ANT Nodes | 9 |
| 5 | ANT Channels | 10 |
| 5.1 | Channel Configuration..... | 10 |
| 5.1.1 | Network..... | 11 |
| 5.1.2 | RF Frequency..... | 11 |
| 5.1.3 | Channel ID | 12 |
| 5.1.4 | Channel Type..... | 12 |
| 5.1.5 | Channel Period..... | 13 |
| 5.1.6 | Data Types & Data Format | 13 |
| 5.1.7 | Example Channel Configuration | 15 |
| 5.2 | Independent Channels | 16 |
| 5.3 | Shared Channels..... | 16 |
| 5.4 | Scanning Channels..... | 17 |
| 5.5 | Establishing a channel..... | 17 |
| 6 | Device Pairing | 18 |
| 6.1 | Pairing Example..... | 18 |
| 7 | ANT Interface | 20 |
| 7.1 | Message Structure | 20 |
| 7.2 | Host MCU Serial Interface – Physical Layer | 20 |
| 7.3 | Host PC Serial Interface | 20 |
| 8 | Example ANT Network Implementation | 20 |
| 8.1 | Implementation using Independent Channels..... | 21 |
| 8.1.1 | Channel between Node B and Node A..... | 23 |
| 8.1.2 | Channel between Node C and Node A..... | 24 |
| 8.1.3 | Channel between Node D and Node A..... | 24 |
| 8.2 | Implementation using Shared Channels | 24 |
| 9 | Appendix A – ANT Message Details | 29 |
| 9.1 | ANT Messages | 29 |
| 9.1.1 | Config Messages | 29 |
| 9.1.2 | Control Messages..... | 29 |
| 9.1.3 | Data Messages..... | 29 |
| 9.1.4 | Channel Event/Response Messages..... | 29 |
| 9.1.5 | Requested Response Messages..... | 29 |
| 9.1.6 | Test Mode | 29 |
| 9.2 | ANT Message Structure - Notes | 29 |
| 9.3 | ANT Message Summary | 30 |
| 9.4 | ANT Product Capabilities | 32 |
| 9.4.1 | Interface | 32 |
| 9.4.2 | Events | 33 |
| 9.5 | ANT Message Details..... | 33 |
| 9.5.1 | ANT Constants | 33 |
| 9.5.2 | Configuration Messages..... | 33 |
| 9.5.3 | Control Messages..... | 42 |
| 9.5.4 | Data Messages..... | 43 |
| 9.5.5 | Channel Response / Event Messages..... | 49 |
| 9.5.6 | Requested Response Messages..... | 52 |
| 9.5.7 | Test Mode | 54 |
| 9.5.8 | Extended Data Messages..... | 55 |
| 9.5.9 | PC Functional Interface Configuration | 61 |
| 9.6 | ANT Library Access | 63 |
| 10 | Appendix B -- SensRcore™ NVM Message Details | 64 |
| 10.1 | SensRcore™ NVM Messages..... | 64 |
| 10.2 | NVM Message Summary..... | 65 |
| 10.2.1 | Serial NVM Messages..... | 66 |



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. ANT's small size and low implementation cost allows effortless integration into the tiny form factor of wrist watches, PDAs, and mobile phones.

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

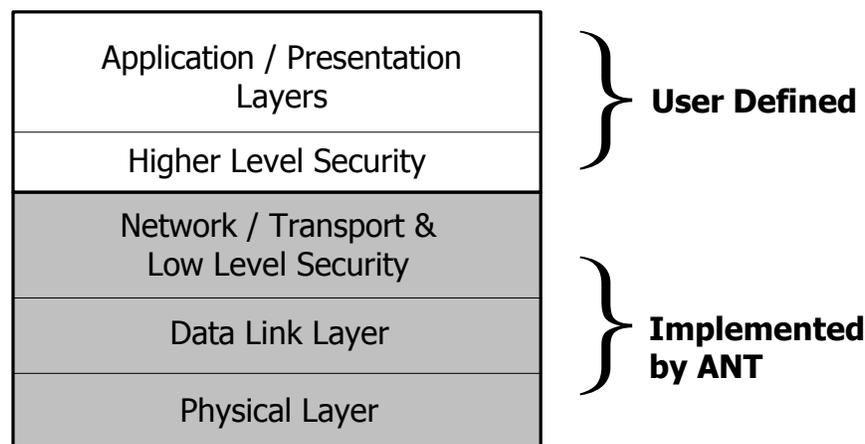


Figure 1-1: OSI Layer model of ANT

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

2 The ANT Product Family

ANT technology has been incorporated into a family of products that allows a particular implementation to be scaled to suit the needs of the application and the vision of the product designer.

ANT technology is available in the following formats.

ANT Single Chip & Chipset

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

1. Nordic Semiconductor 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 nRF24L01). 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. It can be as simple as a 2-node unidirectional connection between a transmitting peripheral device and a receiver, or as complex as a multi-transceiver system with full point-to-multipoint communication capabilities.

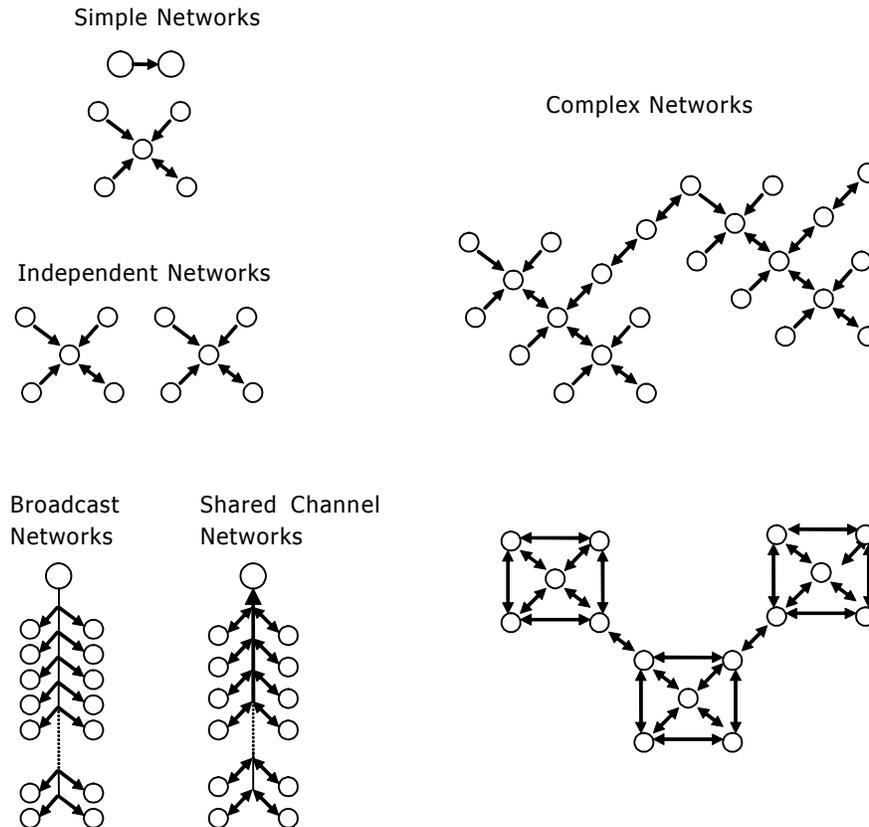


Figure 3-1: Example ANT Networks

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

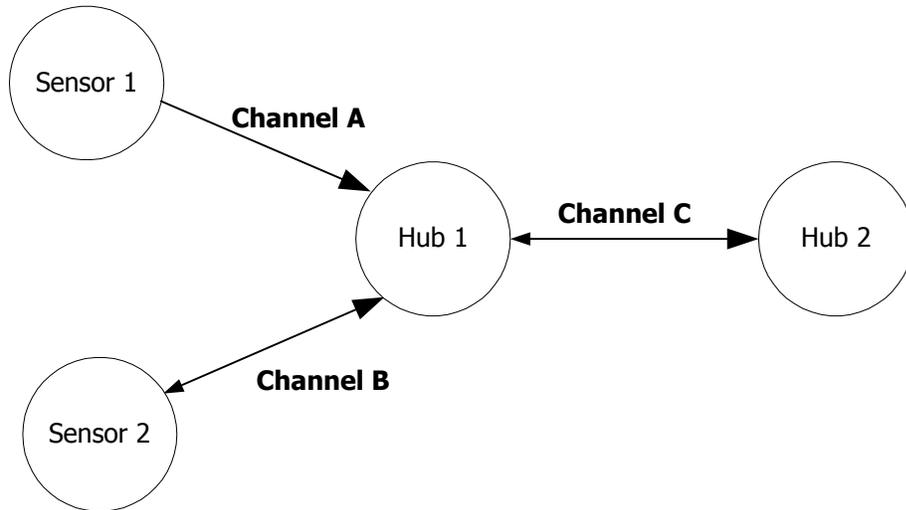


Figure 3-2: A Simple ANT Network

ANT usage and configuration are 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; the slave acts as the primary receiver. In **Figure 3-2**, large arrows indicate the primary data flow from master to slave, with small arrows indicating reverse message flow (e.g. Channel B, C). A channel with a single arrow (e.g. Channel A) is used to represent a one-way link, which supports the use of lower-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.

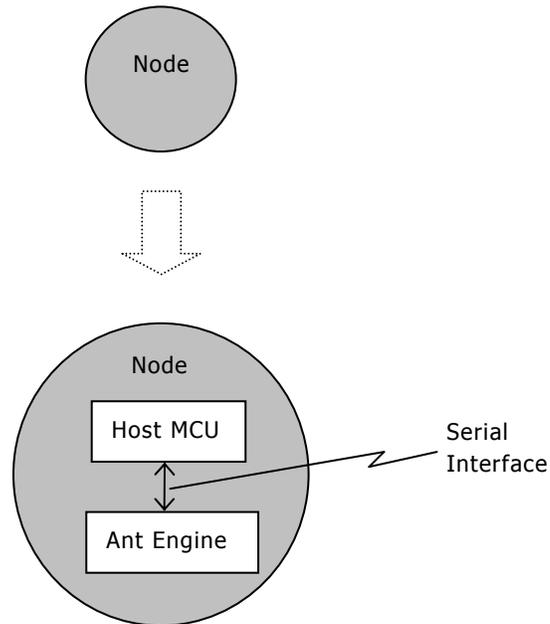


Figure 4-1: Contents of an ANT node

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.

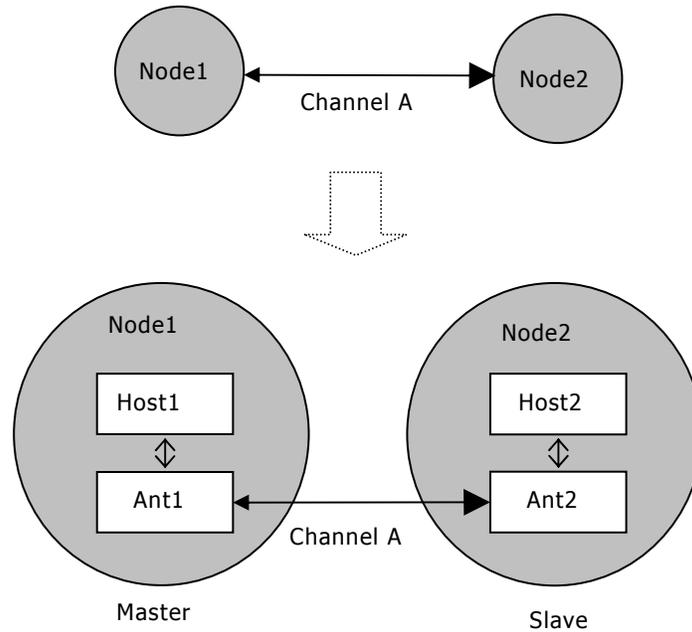


Figure 5-1: Channel communication between two ANT nodes

A channel consists of:

- A master (e.g. Node1)
- A slave (e.g. Node2)

5.1 Channel Configuration

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

- Network
- RF Frequency
- Transmission Type
- 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 that 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. Conversely, a private network could be defined to ensure network privacy and restrict access 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 a network with varied levels of access control and security options. By default ANT firmware assigns the Network Number 0 with the default Public Network Key. This network is open to all participating devices and has no set rules governing its use.

For more information on established public 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.

$$RF_Frequency_val = \frac{Desired_RF_Frequency(MHz) - 2400MHz}{1MHz}$$

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



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

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

5.1.3 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 – Transmission Type, Device Type and Device Number. They are detailed below.

5.1.3.1 Transmission Type

The transmission type is an 8-bit field used to define certain transmission characteristics of a device. Example usage is the SensRcore™ implementation which defines the two least significant bits of the transmission type to define the presence and size of a shared address field at the beginning the data payload, and bit 3 defines the next byte following the address field.

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 the heart rate monitor's data payload will be 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 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 acceptable values in the range of 0 to 255. The values 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 |

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, if a node 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 governs the direction (master to slave or vice versa) and the second 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 communication 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 two-way 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 two-way 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 sides 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. The specific format of the contents of the data payload must be previously established by both host controllers such that data can be properly decoded and interpreted.

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 |
| Transmission Type | 1 | Transmission Type (no shared address) |
| Device Type | 1 | Sample Device Type |
| Channel Type | 0x10 | Bidirectional Transmit Channel |
| Channel Period | 16384 | 2Hz Message Rate |
| Data Type | 0x4E | Broadcast |

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, 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 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. 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 Scanning Channels

Scanning channels are another method that can be used where a single ANT node must receive and possibly process data from many nodes. In this scenario, multiple nodes, setup as transmitters, will broadcast on a single RF frequency to communicate with the central node which receives full time. An example of a scanning channel network would be conceptually similar to that of the shared channel network provided in **Figure 3-1**.

In a scanning channel, the node that is required to establish communication with many other nodes is setup as a receiver that is continuously active. Due to this configuration, this node draws significant power (~18 mA) as its RF is active continuously. The transmit nodes should have unique device numbers in the channel ID so that messages can be correlated to specific devices easily.

5.5 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 pairing mechanism is available. The slave can search for a master with a wild card ID; 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

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

Figure 6-1.

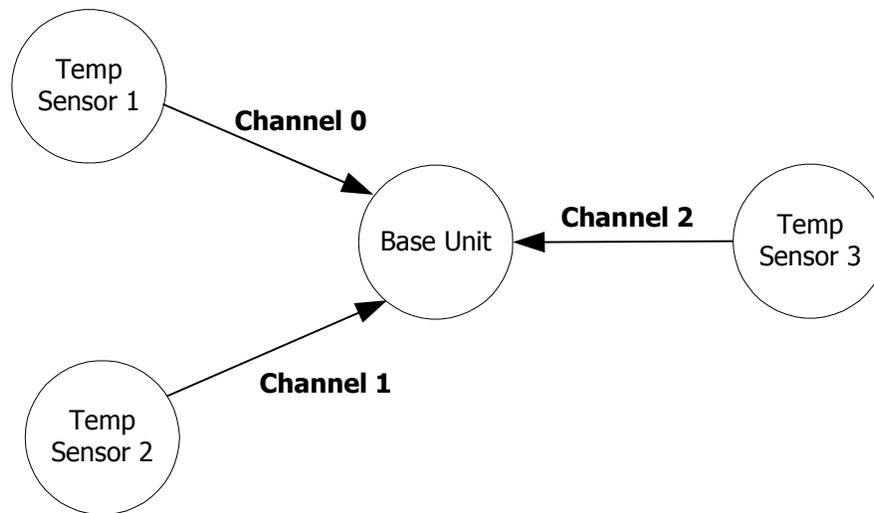


Figure 6-1: Example ANT Network for use in Device Pairing

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

1. Configure Channel
2. Set Channel ID (discoverable)
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 (Transmission Type = 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; the ID 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 required only 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 from the customer.



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.

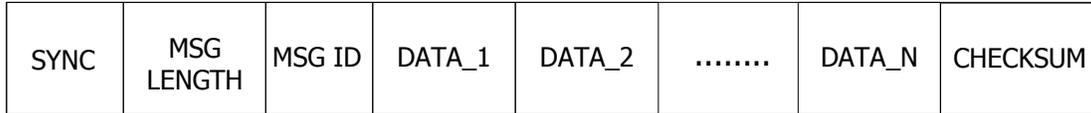


Figure 7-1: ANT Serial Message Structure

As shown above, each message begins with a SYNC byte and ends with a CHECKSUM. The bytes are sent 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. The connection type is selected by the product designer as preferred for the given implementation. The precise details of the physical and electrical interface of each ANT product can be found in each respective ANT product datasheet.

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

A sample network implementation to present the features of the ANT engine is shown in Figure 8-1 below.

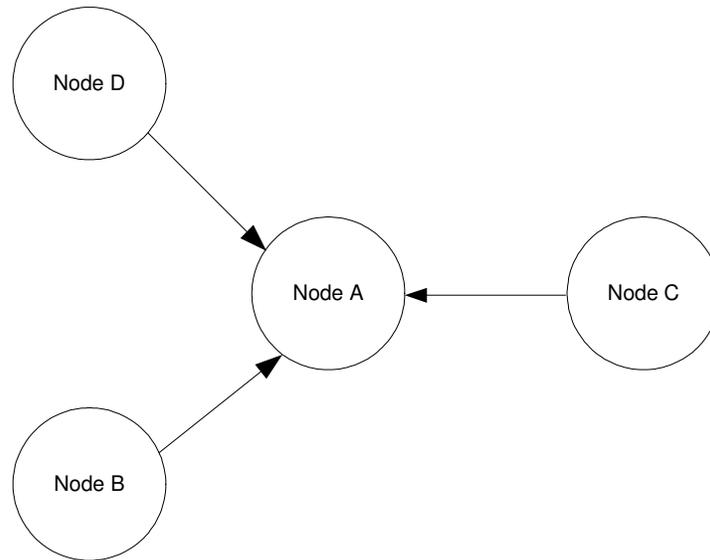


Figure 8-1: Example ANT Network for implementation

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

The following can be assumed.

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

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

8.1 Implementation using Independent Channels

Using independent channels is the simplest method by which the aforementioned network can be implemented. Given the above assumptions, three independent channels are required. The configuration for each channel is shown in the following tables.

Channel between Node B and Node A where Node B will be the master:

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

Channel between Node C and Node A where Node C will be the master:

| Node | Parameter | Value | Description |
|------|-------------------|-------|---------------------------------------|
| C | Network Type | 0 | Default Public Network |
| | RF Frequency | 66 | Default Frequency 2466MHz |
| | Device Number | 10 | Serial Number of Node C |
| | Transmission Type | 1 | Transmission Type (no shared address) |
| | Device Type | 2 | Device Type of Node C |
| | Channel Type | 0x10 | Bidirectional Transmit Channel |
| | Channel Period | 8192 | Default 4Hz Message Rate |
| | Data Type | 0x4F | Acknowledged |
| A | Network Type | 0 | Default Public Network |
| | RF Frequency | 72 | Frequency 2472MHz |
| | Device Number | 10 | Serial Number of Node C |
| | Transmission Type | 1 | Transmission Type (no shared address) |
| | Device Type | 2 | Device Type of Node C |
| | Channel Type | 0x00 | Bidirectional Receive Channel |
| | Channel Period | 8192 | Default 4Hz Message Rate |
| | Data Type | 0x4F | Acknowledged |

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

Section 8.1.1 details the sequence of events and message transactions between the host and ANT for each participating node as the above channels are established and the network is formed. 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

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.

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 step 1.

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

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

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.

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 in the previous step.



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

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

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.

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 in step 1.

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

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 then communicates the success or failure of the transaction back to the host using a message code embedded inside a channel event message. If ANT doesn'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

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.

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 in the previous step.

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

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 are 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 three 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.

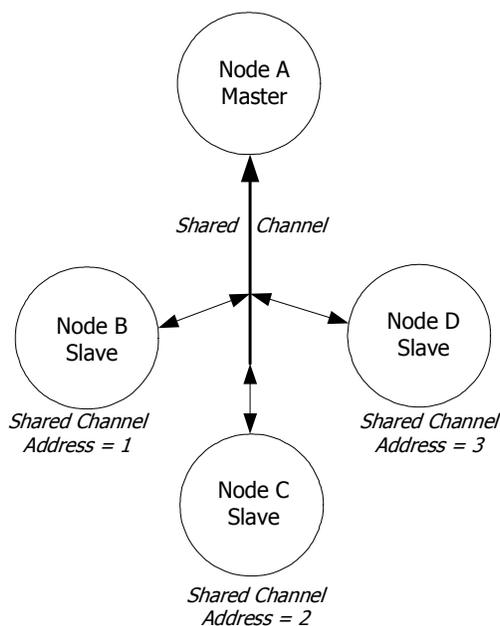


Figure 8-2: Shared channel implementation of sample network

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

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 |
| Transmission Type | 3 | Transmission Type (2 byte shared address) |
| Device Type | 3 | Device Type of Node A |
| Channel Type | 0x20 | Shared Receive Channel |
| Channel Period | 2370 | ~12Hz Message Rate |
| Data Type | 0x4E | Broadcast |

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 |
| Transmission Type | 3 | Transmission Type (2 byte shared address) |
| Device Type | 3 | Device Type of Node A |
| Channel Type | 0x20 | Shared Receive Channel |
| Channel Period | 2370 | ~12Hz Message Rate |
| Data Type | 0x4F | Acknowledged |

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 |
| Transmission Type | 3 | Transmission Type (2 byte shared address) |
| Device Type | 3 | Device Type of Node A |
| Channel Type | 0x20 | Shared Receive Channel |
| Channel Period | 2370 | ~12Hz Message Rate |
| Data Type | 0x4E | Broadcast |

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 |
| Transmission Type | 3 | Transmission Type (2 byte shared address) |
| Device Type | 3 | Device Type of Node A |
| Channel Type | 0x30 | Shared Transmit Channel |
| Channel Period | 2370 | ~12Hz Message Rate |
| Data Type | 0x4E | Broadcast |

Please note:

The Network Type, RF Frequency, Device Number, Transmission Type and Device Type are controlled by the 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 needs 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 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 4-node ANT network using a single shared channel.

Master: Node A

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.

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 in step 1.

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

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

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 the next RF message is addressed.

ANT informs 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 its shared channel address appears.

Slave: Node B

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.

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 in step 1.

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

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

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.

The host will now be informed each time ANT receives a message from the master on the 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

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.

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 in step 1.



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

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

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.

The host will now be informed each time ANT receives a message from the master on the 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 event message from ANT informs the host if an acknowledgement is received from the master or if the data is sent back successfully.

Slave: Node D

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

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



9 Appendix A – ANT Message Details

9.1 ANT Messages

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

9.1.1 Config Messages

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

9.1.2 Control Messages

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

9.1.4 Channel Event/Response Messages

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

9.1.5 Requested Response Messages

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

9.1.6 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.



9.3 ANT Message Summary

| Class | Type | ANT PC Interface Function | Reply | From | Len | Msg ID | Data 1 | Data 2 | Data 3 | Data 4 | Data 5 | Data 6 | Data 7 | Data 8 | Data 9 |
|------------------|------------------------------|--|-------|------|-----|--------|----------------|------------------|----------------|----------------|-------------|------------|--------|--------|--------|
| 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 | Trans. Type | | | | |
| | 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 | | | | | | | |
| | ID List Add | ANT_AddChannelID() | Yes | Host | 6 | 0x59 | Channel Number | Device number | | Device Type ID | Trans. Type | List Index | | | |
| | ID List Config | ANT_ConfigList() | Yes | Host | 3 | 0x5A | Channel Number | List Size | Exclude | | | | | | |
| | Channel Transmit Power | ANT_SetChannelTxPower() | Yes | Host | 2 | 0x60 | Channel Number | TX Power | | | | | | | |
| | Low Priority Search Timeout | ANT_SetLowPriorityChannelSearchTimeout() | Yes | Host | 2 | 0x63 | Channel Number | Search Timeout | | | | | | | |
| | Serial Number Set Channel ID | ANT_SetSerialNumChannelID() | Yes | Host | 3 | 0x65 | Channel Number | Device Type ID | Trans. Type | | | | | | |
| | Enable Ext RX Mesgs | ANT_RxExtMesgsEnable() | Yes | Host | 2 | 0x66 | 0 | Enable | | | | | | | |
| | Enable LED | ANT_EnableLED() | Yes | Host | 2 | 0x68 | 0 | Enable | | | | | | | |
| 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 | | | | | | | | |
| | Open Rx Scan Channel | ANT_OpenRxScanMode() | Yes | Host | 1 | 0x5B | 0 | | | | | | | | |
| | Request Message | ANT_RequestMessage() | Yes | Host | 2 | 0x4D | Channel Number | Message ID | | | | | | | |



| Class | Type | ANT PC Interface Function | Reply | From | Len | Msg ID | Data 1 | Data 2 | Data 3 | Data 4 | Data 5 | Data 6 | Data 7 | Data 8 | Data 9 | |
|-----------------------------|--------------------------|--|-------|----------|-----|--------|--------------------------|----------------|------------------|------------------|--------------------|----------------|--------|--------|--------|--|
| Data 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(-, 0x3E) | - | ANT | 9 | 0x3E | Ver0 | Ver1 | Ver2 | Ver3 | Ver4 | Ver5 | Ver6 | Ver7 | Ver8 | |
| | Capabilities | ->ResponseFunc(-, 0x54) | - | ANT | 6 | 0x54 | Max Channels | Max Networks | Standard Options | Advanced Options | Advanced Options 2 | Max Data Chan. | | | | |
| | Serial Number | ->ResponseFunc(-, 0x61) | - | ANT | 4 | 0x61 | Serial Number | | | | | | | | | |
| Test Mode | CW Init | ANT_InitCWTestMode() | Yes | Host | 1 | 0x53 | 0 | | | | | | | | | |
| | CW Test | ANT_SetCWTestMode() | Yes | Host | 3 | 0x48 | 0 | TX Power | RF Freq | | | | | | | |
| Class | Type | ANT PC Interface Function | Reply | From | Len | Msg ID | Data 1 | Data 2 | Data 3 | Data 4 | Data 5 | Data 6 | Data 7 | Data 8 | Data 9 | |
| Ext Data Messages | Extended Broadcast Data | ANT_SendExtBroadcastData() ->ChannelEventFunc(Chan, EV) | No | ANT/Host | 13 | 0x5D | Channel Number | Device number | | Device Type ID | Trans. Type | Data 6 | Data 7 | Data 8 | Data 9 | |
| | Extended Ack. Data | ANT_SendExtAcknowledgedData() ->ChannelEventFunc(Chan, EV) | No | ANT/Host | 13 | 0x5E | Channel Number | Device number | | Device Type ID | Trans. Type | Data 6 | Data 7 | Data 8 | Data 9 | |
| | Extended Burst Data | ANT_SendExtBurstTransferPacket() ->ChannelEventFunc(Chan, EV) | No | ANT/Host | 13 | 0x5F | Sequence /Channel Number | Device number | | Device Type ID | Trans. Type | Data 6 | Data 7 | Data 8 | Data 9 | |



9.4 ANT Product Capabilities

9.4.1 Interface

| Class | Type | ANT PC Interface Function | nRF24AP1 | ANTAP1MxIB | ANT11TSx3MxIB | |
|-----------------------------|------------------------------|--|-------------------|---------------|---------------|-----|
| Config. Messages | Unassign Channel | ANT_UnAssignChannel() | Yes | Yes | Yes | |
| | Assign Channel | ANT_AssignChannel() | Yes | Yes | Yes | |
| | Channel ID | ANT_SetChannelId() | Yes | Yes | Yes | |
| | Channel Period | ANT_SetChannelPeriod() | Yes | Yes | Yes | |
| | Search Timeout | ANT_SetChannelSearchTimeout() | Yes | Yes | Yes | |
| | Channel RF Frequency | ANT_SetChannelRFFreq() | Yes | Yes | Yes | |
| | Set Network | ANT_SetNetworkKey() | Yes | Yes | Yes | |
| | Transmit Power | ANT_SetTransmitPower() | Yes | Yes | Yes | |
| | ID List Add | ANT_AddChannelID() | No | No | Yes | |
| | ID List Config | ANT_ConfigList() | No | No | Yes | |
| | Channel Transmit Power | ANT_SetChannelTxPower() | No | No | Yes | |
| | Low Priority Search Timeout | ANT_SetLowPriorityChannelSearchTimeout() | No | No | Yes | |
| | Serial Number Set Channel ID | ANT_SetSerialNumChannelID() | No | No | Yes | |
| | Enable Ext RX Mesgs | ANT_RxExtMesgsEnable() | No | No | Yes | |
| | Enable LED | ANT_EnableLED() | No | No | Yes | |
| | Control Messages | SystemReset | ANT_ResetSystem() | Yes | Yes | Yes |
| | | Open Channel | ANT_OpenChannel() | Yes | Yes | Yes |
| Close Channel | | ANT_CloseChannel() | Yes | Yes | Yes | |
| Open Rx Scan Channel | | ANT_OpenRxScanMode() | No | No | Yes | |
| Request Message | | ANT_RequestMessage() | Yes | Yes | Yes | |
| Data Messages | Broadcast Data | ANT_SendBroadcastData() ->ChannelEventFunc(Chan, EV) | Yes | Yes | Yes | |
| | Acknowledge Data | ANT_SendAcknowledgedData() ->ChannelEventFunc(Chan, EV) | Yes | Yes | Yes | |
| | Burst Transfer Data | ANT_SendBurstTransferPacket() ->ChannelEventFunc(Chan, EV) | Yes | Yes | Yes | |
| Channel Event Messages | Channel Response / Event | ->ChannelEventFunc(Chan, MessageCode) or ->ResponseFunc(Chan, MsgID); | Yes | Yes | Yes | |
| Requested Response Messages | Channel Status | ->ResponseFunc(Chan, 0x52) | Yes | Yes | Yes | |
| | Channel ID | ->ResponseFunc(Chan, 0x51) | Yes | Yes | Yes | |
| | ANT Version | ->ResponseFunc(-, 0x3D) | Yes | Yes | Yes | |
| | Capabilities | ->ResponseFunc(-, 0x54) | Yes (4-bytes) | Yes (4-bytes) | Yes (6-bytes) | |
| | Serial Number | ->ResponseFunc(-, 0x61) | No | No | Yes | |
| Test Mode | CW Init | ANT_InitCWTestMode() | Yes | Yes | Yes | |
| | CW Test | ANT_SetCWTestMode() | Yes | Yes | Yes | |
| Ext Data Messages | Extended Broadcast Data | ANT_SendExtBroadcastData() ->ChannelEventFunc(Chan, EV) | No | No | Yes | |
| | Extended Acknowledge Data | ANT_SendExtAcknowledgedData() ->ChannelEventFunc(Chan, EV) | No | No | Yes | |
| | Extended Burst Data | ANT_SendExtBurstTransferPacket() ->ChannelEventFunc(Chan, EV) | No | No | Yes | |

9.4.2 Events

See Section 9.5.5.1 for Event details.

| Name | nRF24AP1 | ANTAP1MxIB | ANT21Tx3MxIB |
|--------------------------------|----------|------------|--------------|
| RESPONSE_NO_ERROR | Yes | Yes | Yes |
| EVENT_RX_SEARCH_TIMEOUT | Yes | Yes | Yes |
| EVENT_RX_FAIL | Yes | Yes | Yes |
| EVENT_TX | Yes | Yes | Yes |
| EVENT_TRANSFER_RX_FAILED | Yes | Yes | Yes |
| EVENT_TRANSFER_TX_COMPLETED | Yes | Yes | Yes |
| EVENT_TRANSFER_TX_FAILED | Yes | Yes | Yes |
| EVENT_CHANNEL_CLOSED | Yes | Yes | Yes |
| EVENT_RX_FAIL_GO_TO_SEARCH | No | Yes | Yes |
| EVENT_CHANNEL_COLLISION | No | Yes | Yes |
| EVENT_TRANSFER_TX_START | No | No | Yes |
| CHANNEL_IN_WRONG_STATE | Yes | Yes | Yes |
| CHANNEL_NOT_OPENED | Yes | Yes | Yes |
| CHANNEL_ID_NOT_SET | Yes | Yes | Yes |
| CLOSE_ALL_CHANNELS | No | No | Yes |
| TRANSFER_IN_PROGRESS | Yes | Yes | Yes |
| TRANSFER_SEQUENCE_NUMBER_ERROR | Yes | Yes | Yes |
| TRANSFER_IN_ERROR | No | No | Yes |
| INVALID_MESSAGE | Yes | Yes | Yes |
| INVALID_NETWORK_NUMBER | Yes | Yes | Yes |
| INVALID_LIST_ID | No | No | Yes |
| INVALID_SCAN_TX_CHANNEL | No | No | Yes |
| NVM_FULL_ERROR | No | No | Yes |
| NVM_WRITE_ERROR | No | No | Yes |

9.5 ANT Message Details

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

9.5.1 ANT Constants

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

1. MAX_CHAN – number of supported channels. Valid channels are 0..(MAX_CHAN-1).
2. MAX_NET – number of supported networks. Valid networks are 0..(MAX_NET-1).

These values can be determined for the specific ANT implementation by requesting the capability message.

9.5.2 Configuration Messages

The following messages are used to configure a channel. Care should be taken to configure all appropriate pieces of information for a channel before opening it. All configuration commands return a response to



indicate their success or failure. Therefore, a simple state machine can be setup for configuration of channels that advances states only when a RESPONSE_NO_ERROR is received for the current command and to re-send upon failures.

9.5.2.1 Unassign Channel (0x41)

```
BOOL ANT_UnAssignChannel(UCHAR ucChannel);
```

| Parameters | Type | Range | Description |
|--|-------|-----------------|-------------------------------|
| Channel Number | UCHAR | 0..MAX_CHAN - 1 | The channel to be unassigned. |
| <pre>// Example usage ANT_AssignChanel(0, 0x00, 0); .. ANT_UnAssignChannel(0);</pre> | | | |

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

9.5.2.2 Assign Channel (0x42)

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

| 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. |
| <pre>// Example Usage ANT_AssignChannel(0, 0x00, 0); // receive channel on network number 0</pre> | | | |

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.

9.5.2.3 Channel ID (0x51)

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

| Parameters | Type | Bit Range | Range | Description |
|-------------------|---------------------------|-----------|---------------|---|
| Channel Number | UCHAR | - | 0..MAX_CHAN-1 | The channel number |
| Device Number | USHORT (little endian) | - | 0..65535 | The device number. For a 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. |
| Transmission Type | UCHAR | - | 0..255 | The transmission type. For a receiver use 0 to receive from any transmission type. |

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

This message configures the channel ID 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 the ID 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 Transmission Type). 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 Transmission Type. 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 i.e. if you know the exact device you are looking for, then pairing is irrelevant.

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



9.5.2.4 Channel Messaging Period (0x43)

BOOL ANT_SetChannelPeriod(UCHAR ucChannel, USHORT usMessagePeriod);

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

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

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

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

Note: If the 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.

9.5.2.5 Channel Search Timeout (0x44)

BOOL ANT_SetChannelSearchTimeout(UCHAR ucChannelNum, UCHAR ucSearchTimeout);



| Parameters | Type | Range | Default | Description |
|----------------|-------|---------------|-------------------|---|
| Channel Number | UCHAR | 0..MAX_CHAN-1 | - | The channel number |
| Search Timeout | UCHAR | 0..255 | 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 of time that the receiver will search for a channel before timing out.

*In new modules, the value of 255 sets an infinite search time-out as opposed to a search timeout of 10.5 minutes.

9.5.2.6 Channel RF Frequency (0x45)

```
BOOL ANT_SetChannelRFFreq(UCHAR ucChannel, UCHAR ucRFFreq);
```

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

// Example Usage

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

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

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.

9.5.2.7 Set Network Key(0x46)

```
BOOL ANT_SetNetworkKey(UCHAR ucNetNumber, UCHAR *pucKey);
```



| Parameters | Type | Range | Description |
|----------------|-------|--------------|--------------------|
| Network Number | UCHAR | 0..MAX_NET-1 | The network number |
| Network Key 0 | UCHAR | 0..255 | Network byte 0 |
| Network Key 1 | UCHAR | 0..255 | Network byte 1 |
| Network Key 2 | UCHAR | 0..255 | Network byte 2 |
| Network Key 3 | UCHAR | 0..255 | Network byte 3 |
| Network Key 4 | UCHAR | 0..255 | Network byte 4 |
| Network Key 5 | UCHAR | 0..255 | Network byte 5 |
| Network Key 6 | UCHAR | 0..255 | Network byte 6 |
| Network Key 7 | UCHAR | 0..255 | Network byte 7 |

```
// Example Usage
UCHAR aucNetworkKey = {0x00, 0x01, 0x00, 0x01, 0x00, 0x01, 0x00, 0x01}; // sample Network Key

ANT_SetNetworkKey(1, aucNetworkKey); // assign the network key to network number 1
// wait for RESPONSE_NO_ERROR
ANT_AssignChannel(0, 0x00, 1); // receive channel on network 1
```

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

This command is not required when using the default public network. The default public network key is already assigned by default to Network Number 0.

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

9.5.2.8 Transmit Power (0x47)

```
BOOL ANT_SetTransmitPower(UCHAR ucTransmitPower);
```

| Parameters | Type | Range | Default | Description |
|----------------|-------|-------|-------------|---|
| Filler | UCHAR | 0 | 0 | A filler 0 byte that must be included |
| Transmit Power | UCHAR | 0..3 | 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.

9.5.2.9 Add List ID (0x59)

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

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

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

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

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

This message is sent to the module to add channel IDs to the inclusion/exclusion list. When this list is used, these ID's will either be the only IDs accepted in a wild card search or ID's that will not be discovered at all.

The use of these ID's is enabled by the ListConfig command detailed below. A maximum of 4 IDs can be placed in the list.

9.5.2.10 Config List ID (0x5A)

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

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

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

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

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



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

9.5.2.11 Set Channel Tx Power (0x60)

```
BOOL ANT_SetChannelTxPower(UCHAR ucChannel, UCHAR ucTxPower);
```

| Parameters | Type | Range | Description |
|----------------|-------|---------------|---|
| Channel Number | UCHAR | 0..MAX_CHAN-1 | The channel number |
| Transmit Power | UCHAR | 0..3 | 0 = TX Power -20 dBm 1 = TX Power -10 dBm 2 = TX Power -5 dBm 3 = TX Power 0 dBm |

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

This message is sent to the module to set the transmit power level for a specified channel.

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

9.5.2.12 Channel Low Priority Search Timeout (0x63)

```
BOOL ANT_SetLowPriorityChannelSearchTimeout(UCHAR ucChannelNum, UCHAR ucSearchTimeout);
```

| Parameters | Type | Range | Default | Description |
|----------------|-------|---------------|-------------------|---|
| Channel Number | UCHAR | 0..MAX_CHAN-1 | - | The channel number |
| Search Timeout | UCHAR | 0..255 | 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. A value of 255 specifies infinite search timeout. 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_SetLowPriorityChannelSearchTimeout(0, 24); // low priority search timeout is 60s
```

This message is sent to the module to configure the length of time that the receiver will search for a channel in low priority mode before timing out and switching to a high priority search. Low priority mode search differs from high priority mode search in the fact that it does not interrupt other channels running on the device to search as high priority search does. When the low-priority search times out, it drops to high priority search until either the device is found or the high priority search times out as well.

9.5.2.13 Serial Number Channel ID (0x65)

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

| Parameters | Type | Bit Range | Range | Description |
|-------------------|---------------|-----------|---------------|---|
| Channel Number | UCHAR | - | 0..MAX_CHAN-1 | The channel number |
| Pairing Request | UCHAR (1bit) | 7 | 0..1 | Pairing Request. Set this bit on 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. |
| Transmission Type | UCHAR | - | 0..255 | The transmission type. For a receiver, use 0 to receive from any transmission type. |

```
// Example Usage
// Tx channel
ANT_AssignChannel(0, 0x10, 0);
// wait for RESPONSE_NO_ERROR
ANT_SetSerialNumChannelId(0, 120, 123);
/*****/
// Rx channel
ANT_AssignChannel(0, 0x00, 0);
// wait for RESPONSE_NO_ERROR
ANT_SetSerialNumChannelId(0, 120, 123); // device number is wild-card
/*****/
// Pairing bit on Rx channel
ANT_AssignChannel(0, 0x00, 0);
// wait for RESPONSE_NO_ERROR
ANT_SetSerialNumChannelId(0, 248, 123); // device number is wild-card, device type 120 with pairing bit ON
```

This message configures the channel ID to be used by a specific channel in the same way as the ChannelID command (see 9.4.2.3) only it uses the two least significant bytes of the device's serial number as the device number.

9.5.2.14 Enable Extended Messages (0x66)

BOOL ANT_RxExtMesgsEnable (UCHAR ucEnable);

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

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

This message is sent to the module to enable or disable the extended Rx messages on the module. When this setting is enabled, received messages contain an additional four bytes that precede the normal data bytes. The channel ID should match that of the master.

9.5.2.15 Enable LED (0x68)

BOOL ANT_EnableLED(UCHAR ucEnable);



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

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

This message is sent to the module to enable or disable the LED on the module. When the LED is enabled, it will blink each time a RF transmit or a RF receive event is detected by the module.

9.5.3 Control Messages

9.5.3.1 Reset System(0x4A)

```
BOOL ANT_ResetSystem(void);
```

| Parameters | Type | Range | Description |
|------------|-------|-------|-------------|
| Filler | UCHAR | 0 | |

This message is sent to the module to reset the system and to put it in a known low-power state. Execution of this command terminates all channels. All information previously configured in the system can no longer be considered valid.

9.5.3.2 Open Channel (0x4B)

```
BOOL ANT_OpenChannel(UCHAR ucChannel);
```

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

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

9.5.3.3 Close Channel (0x4C)

```
BOOL ANT_CloseChannel(UCHAR ucChannel);
```

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

This message is sent to close a channel that has been previously opened. When a channel is closed it remains assigned with all associated parameters still valid. The channel may be reopened at any time with the Open Channel Command.

9.5.3.4 Request Message (0x4D)

```
BOOL ANT_RequestMessage(UCHAR ucChannel, UCHAR ucMessageID);
```

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

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

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

Valid messages include Channel Status, Channel ID, ANT Version, and Capabilities. Requesting one of these messages causes ANT to send the appropriate response message. Please see these messages for specific details.

9.5.3.5 Open Rx Scan Channel (0x5B)

```
BOOL ANT_OpenRxScanChannel(UCHAR ucChannel);
```

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

```
// Example Usage
ANT_OpenRxScanChannel (0); // open scanning channel
ANT_SendExtBroadcastData(Channel_1, DATA); // send data to be transmitted to
```

This message is sent to the module to open a channel that has been previously assigned and configured as a receive channel with the configuration messages. Execution of this command causes the channel to commence operation in scanning mode. In scanning mode, the radio is active in receive mode 100% of the time so no other channels but the scanning channel can run. The scanning channel picks up any message regardless of period that is being transmitted on its RF frequency and matches its channel ID mask. It can receive from multiple devices simultaneously. It can also have messages pending to be sent to MAX_CHAN – 1 individual devices that are communicating with the scanning device. This is achieved by passing an extended data message with the correct Channel ID for the device the data is to be sent to on a channel in the range of 1:MAX_CHAN – 1.

9.5.4 Data Messages

There are three methods for sending and receiving data on a channel. These methods are described below.

9.5.4.1 Broadcast Data (0x4E)

```
BOOL 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 sends a message to the receiver at the programmed channel period. After every transmission ANT generates 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 re-transmitted. It is up to the transmit MCU on every message to send new data.

Receiver

The corresponding channel receiver receives 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, then sends a special library-only event EVENT_RX_BROADCAST to let you know that a valid broadcast message has been received.

9.5.4.2 Acknowledged Data (0x4F)

```
BOOL 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_SendAcknowledgedData(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;
    }
}

```

The Acknowledged Data message can be used in place of the Broadcast Data message to ensure the successful transmission of data. Acknowledged data is transmitted in the same transmission time slot as Broadcast Data but extends the length of the timeslot to accommodate the acknowledgement. Acknowledged Data transmissions cannot be originated from channels configured for transmit only.

Transmitter

The transmitter behaves as 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 from the transmitter causes an Acknowledged Data message to be sent to the receiver MCU. If the message reception fails, an EVENT_RX_FAIL occurs.

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

9.5.4.3 Burst Data (0x50)

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

```
BOOL ANT_SendBurstTransferPacket(UCHAR ucChannelSeq, UCHAR* pucData); // 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 is 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 from the transmitter causes 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 occurs.

Sequence Numbers

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

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

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

Example:

Channel = 3

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

9.5.5 Channel Response / Event Messages

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

9.5.5.1 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)

* Not all message Events are generated by all products. See section 9.4.2 for information on which event messages are supported by which products.

| 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 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_FAIL_GO_TO_SEARCH | 8 | The channel has dropped to search after missing too many messages. |
| EVENT_CHANNEL_COLLISION | 9 | Two channels have drifted into each other and overlapped in time on the device causing one channel to be blocked. |
| EVENT_TRANSFER_TX_START | 10 | Sent when a burst transfer begins after, effectively on the next message period after the burst transfer message has |

| Name | Value | Description |
|--------------------------------|-------|---|
| | | been sent to the device. |
| CHANNEL_IN_WRONG_STATE | 21 | Returned on attempt to perform an action on a channel that is not valid for the channel's state |
| CHANNEL_NOT_OPENED | 22 | 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 |
| CLOSE_ALL_CHANNELS | 25 | Returned when an OpenRxScan() command is sent while other channels are open. |
| TRANSFER_IN_PROGRESS | 31 | Returned on an attempt to communicate on a channel with a transmit transfer in progress. |
| TRANSFER_SEQUENCE_NUMBER_ERROR | 32 | Returned when sequence number is out of order on a Burst Transfer |
| TRANSFER_IN_ERROR | 33 | Returned when a burst message passes the sequence number check but will not be transmitted. |
| INVALID_MESSAGE | 40 | Returned when message has invalid parameters |
| INVALID_NETWORK_NUMBER | 41 | Returned when an invalid network number is provided. As mentioned earlier, valid network numbers are between 0 and MAX_NET-1. |
| INVALID_LIST_ID | 48 | Returned when the provided list ID or size exceeds the limit. |
| INVALID_SCAN_TX_CHANNEL | 49 | Returned when attempting to transmit on ANT channel 0 in scan mode. |
| NVM_FULL_ERROR | 64 | Returned when the NVM for SensRcore mode is full. |
| NVM_WRITE_ERROR | 65 | Returned when writing to the NVM for SensRcore mode fails. |

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



| Name | Value | Description |
|---|-------|-------------|
| <pre> // data packet was lost Break; } Break; Case Default: // catch unexpected message codes Break; } } </pre> | | |

9.5.6 Requested Response Messages

The following messages are returned in response to Request Message 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.

9.5.6.1 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 MSG_CHANNEL_STATUS_ID:
            Switch (aucResponseBuffer[1]) // channel status
            {
                Case 0:
                    // channel is un-assigned
                    Break;
                Case 1:
                    // channel is assigned
                    Break;
                ..
            }
            Break;
    }
}

```

9.5.6.2 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 |
| Transmission Type | UCHAR | 0..255 | The transmission type |

This message returns the channel ID of the specified channel. This message is useful when trying to pair devices. When a receiver is attempting to pair to a transmitter, it will typically set one or more of the Device Number, Device Type ID, or Transmission Type 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 to search for this specific device. See pairing under the Usage section for more details.

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

9.5.6.3 ANT Version (0x3E)

ResponseFunc (-, 0x3E)

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 |

9.5.6.4 Capabilities (0x54)

ResponseFunc (-, 0x54)

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



| Parameters | Type | Range | Description |
|---|-------|------------------|---|
| Max ANT Channels | UCHAR | 0..MAX_CHAN | Returns the Number of ANT channels available |
| Max Networks | UCHAR | 0..MAX_NET-1 | Returns the number of networks available |
| Standard Options | UCHAR | 0..255 | The Standard Options bitfield is encoded as follows: <pre>#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</pre> Other bits are reserved |
| Advanced Options | UCHAR | 0..255 | The Advanced Options bitfield is encoded as follows: <pre>#define CAPABILLITES_NETWORK_ENABLED 0x02 #define CAPABILITIES_SERIAL_NUMBER_ENABLED 0x08 #define CAPABILITEIS_PER_CHANNEL_TX_POWER_ENABLED 0x10 #define CAPABILITIES_LOW_PRIORITY_SEARCH_ENABLED 0x20 #define CAPABILLITES_SENSRCORE_ENABLED 0x40 #define CAPABILLITES_SEARCH_LIST_ENABLED 0x80</pre> Other bits are reserved |
| Advanced Options 2 *(Only available in new versions) | UCHAR | 0..255 | The Advanced Options 2 bitfield is encoded as follows: <pre>#define CAPABILITIES_LED_ENABLED 0x01 #define CAPABILITIES_EXT_MESSAGE_ENABLED 0x02 #define CAPABILITIES_SCAN_CHANNEL_ENABLED 0x04</pre> Other bits are reserved |
| Max Data Channels | UCHAR | 0..MAX_DATA_CHAN | Returns the number of data channels available |

9.5.6.5 Device Serial Number (0x61)

ResponseFunc (-, 0x61)

The serial number is a 4-byte, little-endian encoded unsigned integer. Please note that this message is not supported on all ANT products, especially the nRF24AP1.

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

9.5.7 Test Mode

9.5.7.1 Init CW Test Mode (0x53)

BOOL ANT_InitCWTestMode(void);

| Parameters | Type | Range | Description |
|------------|-------|-------|-------------|
| Filler | UCHAR | 0 | |

This function must be called before the CW Test Mode message below, 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.

9.5.7.2 CW Test Mode (0x48)

```
BOOL ANT_SetCWTestMode(UCHAR ucTransmitPower, UCHAR ucRFChannel);
```

| Parameters | Type | Range | Description |
|--|-------|--------|---|
| Filler | UCHAR | 0 | A filler 0 byte that must be included |
| Transmit Power | UCHAR | 0..3 | 0 = TX Power -20 dB 1 = TX Power -10 dB 2 = TX Power -5 dB 3 = TX Power 0 dB |
| Channel RF Frequency | UCHAR | 0..127 | Channel Frequency = 2400 MHz + Channel RF Frequency Number * 1.0 MHz |
| <pre>// Example Usage ANT_InitCWTestMode(); // wait for RESPONSE_NO_ERROR ANT_SetCWTestMode(3, 57); // set RF power to 0dBm and CW 2457MHz</pre> | | | |

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

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.

9.5.8 Extended Data Messages

Each of the Data Message functions described in section 9.4.4 can be used

9.5.8.1 Extended Broadcast Data (0x5D)

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

or

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

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

```
// Example Usage

// Transmitter
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
    Switch (ucEvent)
    {
        Case EVENT_TX:
            Switch (ucChannel)
            {
                Case Channel_0:
                    ANT_SendExtBroadcastData(Channel_0, DATA);
                    Break;
                ..
            }
            Break;
    }
}

/*****
// Receiver
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
    Switch (ucEvent)
    {
        Case EVENT_RX_EXT_BROADCAST: // PC applications only; use MsgID 0x5D in embedded
            Switch (ucChannel)
            {
                Case Channel_0:
                    // process received data which is in channel event buffer
                    Break;
                ..
            }
            Break;
    }
}
*****/
```

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

Receiver

The corresponding channel receiver receives the data at its programmed channel period and generate an Extended 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, then sends a special library-only event EVENT_RX_EXT_BROADCAST to let you know that a valid extended broadcast message has been received.

9.5.8.2 Extended Acknowledged Data (0x5E)

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

or

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



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

```
// Example Usage
// Transmitter
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
    Switch (ucEvent)
    {
        Case EVENT_TRANSFER_TX_COMPLETED:
            Switch (ucChannel)
            {
                Case Channel_0:
                    ANT_SendExtAcknowledgedData(Channel_0, DATA);
                    Break;
                ..
            }
            Break;
    }
}

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

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

Receiver

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

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

9.5.8.3 Extended Burst Data (0x5F)

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

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

or

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



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

```
// Example Usage
// Transmitter
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
    Switch (ucEvent)
    {
        Case EVENT_TRANSFER_TX_COMPLETED:
            Switch (ucChannel)
            {
                Case Channel_0:
                    ANT_SendExtBurstData(Channel_0, DATA, 4); // 8 bytes per packet, 32 bytes total
                    Break;
                ..
            }
            Break;
    }
}
/*****/
// Receiver
BOOL ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent)
{
    Switch (ucEvent)
    {
        Case EVENT_RX_EXT_BURST_PACKET: // PC applications only; use MsgID 0x5F in embedded
            Switch (ucChannel)
            {
                Case Channel_0:
                    // process received data which is in channel event buffer one packet at a time validating the
                    // sequence
                    Break;
                ..
            }
            Break;
    }
}
}
```

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

Receiver

Reception of Burst Data from the transmitter causes Extended 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 occurs.

9.5.9 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 are exchanged directly through a serial interface.

9.5.9.1 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.

9.5.9.2 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.

9.5.9.3 ANT_Close

```
void ANT_Close (void);
```

| Parameters | Type | Range | Description |
|------------|------|-------|-------------|
| None | | | |

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

ANT_Close closes the USB connection to the ANT module.



9.5.9.4 ANT_AssignResponseFunction

```
void ANT_AssignResponseFunction(RESPONSE_FUNC pfResponse, UCHAR *pucResponseBuffer);
```

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

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

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

The response function must be a C function.

9.5.9.5 ANT_AssignChannelEventFunction

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

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

```
// Example Usage
BOOL ANT_ChannelEventFunction(UCHAR ucChannel, UCHAR ucEvent);
UCHAR aucChannelEventBuffer[MESG_DATA_SIZE];
.
.
ANT_AssignChannelEventFunction(channel_0, &ANT_ChannelEventFunction, aucChannelEventBuffer);
```

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

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

9.6 ANT Library Access

As part of the development kit, developers are able to utilize the ANT Library functions described in Section 0. 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 ANT 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 0 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");

        if(ANT_Close == NULL){
            FreeLibrary(hDLL);
            return;
        }
    }
}
```

10 Appendix B -- SensRcore™ NVM Message Details

10.1 SensRcore™ NVM Messages

The NVM configuration messages allow the Host to program the NVM with a desired boot configuration. Further details are included in D00000781 ANT SensRcore™ Messaging and Usage.



10.2 NVM Message Summary

| Class | Type | ANT PC Interface Function | Reply | From | Len | Msg ID | Data 1 | Data 2 | Data 3 | Data 4 | Data 5 | Data 6 | Data 7 | Data 8 | Data 9 | Data 10 |
|---------------------|------------------------|----------------------------|-------|------|------|--------|--------|------------|----------------|------------|------------|------------|------------|------------|------------|------------|
| Serial NVM Messages | NVM Write | ANT_NVM_Write() | Yes | Host | 2-10 | 0x56 | Cmd ID | Cmd Data 1 | Cmd Data 2 | Cmd Data 3 | Cmd Data 4 | Cmd Data 5 | Cmd Data 6 | Cmd Data 7 | Cmd Data 8 | Cmd Data 9 |
| | NVM Clear | ANT_NVM_Clear() | Yes | Host | 3 | 0x57 | 0x00 | 0x00 | 0x00 | | | | | | | |
| | NVM Dump | ANT_NVM_Dump() | Yes | Host | 3 | 0x57 | 0x00 | 0x01 | 0x00 | | | | | | | |
| | NVM Set Default Sector | ANT_NVM_SetDefaultSector() | Yes | Host | 3 | 0x57 | 0x00 | 0x02 | Default Sector | | | | | | | |
| | NVM End Sector | ANT_NVM_EndSector() | Yes | Host | 3 | 0x57 | 0x00 | 0x03 | 0x00 | | | | | | | |
| | NVM Lock | ANT_NVM_Lock() | Yes | Host | 3 | 0x57 | 0x00 | 0x05 | 0x00 | | | | | | | |



10.2.1 Serial NVM Messages

10.2.1.1 NVM_Write (0x56)

```
void ANT_NVM_Write(UCHAR ucSize, UCHAR *pucCmd);
```

| Parameters | Type | Range | Description |
|----------------|-------|--------|---|
| Size | UCHAR | 0..10 | The size of the command, which is the number of data bytes + 1 for the Cmd ID |
| Cmd ID | UCHAR | 0..255 | The ID of the command to be programmed into NVM |
| Command Data 0 | UCHAR | 0..255 | Command byte 0 |
| Command Data 1 | UCHAR | 0..255 | Command byte 1 |
| Command Data 2 | UCHAR | 0..255 | Command byte 2 |
| Command Data 3 | UCHAR | 0..255 | Command byte 3 |
| Command Data 4 | UCHAR | 0..255 | Command byte 4 |
| Command Data 5 | UCHAR | 0..255 | Command byte 5 |
| Command Data 6 | UCHAR | 0..255 | Command byte 6 |
| Command Data 7 | UCHAR | 0..255 | Command byte 7 |
| Command Data 8 | UCHAR | 0..255 | Command byte 8 |

```
// Example Usage
UCHAR aucCommand = {0x42, 0x00, 0x10, 0x00}; // sample Network Key

ANT_NVMWrite(4, aucCommand); // write aucCommand, an assign channel command that assigns channel 0 as
// a Tx Channel
```

This message writes a command into NVM. These commands are then read out and executed in order when the module starts up in SensRcore™ mode. This command can be of variable length and requires only as many data bytes as are required by the command that is being programmed into NVM.

10.2.1.2 NVM_Clear (0x57)

```
void ANT_NVM_Clear();
```

| Parameters | Type | Range | Default | Description |
|------------|-------|-------|---------|---------------------------------------|
| Filler | UCHAR | 0 | 0 | A filler 0 byte that must be included |
| Cmd Number | UCHAR | 0 | 0 | Specifies the clear command |
| Filler | UCHAR | 0 | 0 | A filler 0 byte that must be included |

```
// Example Usage
ANT_NVM_Clear(); // clears the NVM
```

This message is sent to the module to clear the NVM. This command must be performed before commands are written into the NVM using NVM_Write.

10.2.1.3 NVM_Dump (0x57)

```
void ANT_NVM_Dump();
```

| Parameters | Type | Range | Range | Description |
|------------|-------|-------|-------|---------------------------------------|
| Filler | UCHAR | 0 | 0 | A filler 0 byte that must be included |
| Cmd Number | UCHAR | 1 | 1 | Specifies the dump command |
| Filler | UCHAR | 0 | 0 | A filler 0 byte that must be included |

```
// Example Usage
/*****/
ANT_NVM_Dump();
// NVM_Write commands will be sent to the Response func and it will end with a message of the format
// [57][00][04][xx] where [xx] is the number of messages that have been read out
```

This message is sent to the module to dump the contents of the NVM. It causes the part to send NVM_Write commands to the Host with the data commands that have been sent in, unless the NVM has been locked from reading (**Error! Reference source not found.**). It also will send out NVM_EndSector commands to denote the end of a sector and an NVM_DefaultSector command with the stored default sector. Finally, an NVM_Cmd with a command number of 0x04 will be sent to signify the end of the Dump and denote how many commands were sent.

10.2.1.4 NVM_SetDefaultSector (0x57)

```
void ANT_NVM_SetDefaultSector(UCHAR ucDefaultSector);
```

| Parameters | Type | Range | Default | Description |
|----------------|-------|--------|---------|--|
| Filler | UCHAR | 0 | 0 | A filler 0 byte that must be included |
| Cmd Number | UCHAR | 2 | 2 | Specifies the set default sector command |
| Default Sector | UCHAR | 0..255 | 0 | The default sector to be set |

```
// Example Usage
ANT_NVM_SetDefaultSector(3); // set the Default Sector to 3
```

This message is sent to the module to set the default sector. This sector is what will be executed after sector 0 is complete. Only one sector besides sector 0 is ever executed on startup. The default sector is not used if the Read Pins for Sector command is present in sector 0, or the default sector is set to 0, which is the default.

10.2.1.5 NVM_EndSector (0x57)

```
void ANT_NVM_EndSector();
```

| Parameters | Type | Range | Default | Description |
|------------|-------|-------|---------|---------------------------------------|
| Filler | UCHAR | 0 | 0 | A filler 0 byte that must be included |
| Cmd Number | UCHAR | 3 | 3 | Specifies the end sector command |
| Filler | UCHAR | 0 | 0 | A filler 0 byte that must be included |

```
// Example Usage
ANT_NVM_EndSector(); // puts a sector break in the NVM
```

This message is sent to the module to put a sector break in the NVM image. Sector breaks are used to separate optional blocks of configuration commands. The first sector (sector 0) is always run on startup so a typical image would feature generic commands in this sector. By adding more sectors, different devices using the same NVM image can be made to have different configurations using separate sectors that may be selected with the DevSel pins.

10.2.1.6 NVM_Lock (0x57)

```
void ANT_NVM_Lock();
```



| Parameters | Type | Range | Default | Description |
|------------|-------|-------|---------|---------------------------------------|
| Filler | UCHAR | 0 | 0 | A filler 0 byte that must be included |
| Cmd Number | UCHAR | 5 | 5 | Specifies the lock command |
| Filler | UCHAR | 0 | 0 | A filler 0 byte that must be included |

```
// Example Usage  
ANT_NVM_Lock(); // Locks the NVM
```

This message is sent to the module to lock the NVM. This command prevents the contents of the NVM from being dumped in order to provide security for NVM images. The only way to Dump the contents after an NVM_Lock command is to clear the NVM, and rewrite it.