ObservANT User's Guide

Copyright Information and Usage Notice

Revision History

Revision	Effective Date	Description
1.0	December 2014	Initial release
1.1	February 2016	Update to include information about formatting values as ASCII characters

Table of Contents

List of Figures

List of Tables

1 Introduction

ObservANT is an ANT debugging tool which can be used to receive and decode specially formatted 'debug messages' sent from ANT enabled devices.

Information contained in debug messages may include: the device's current state, the number of channels open, the value of a counter, the contents of specific variable, etc. Debug messages are received by ObservANT which displays and logs the information in a convenient way. The process is analogous to inserting print statements in order to monitor an application when a debugger is unavailable.

2 Channel Configuration

2.1 Topology

The debug channel topology is very simple. The device under test will always act as the master and ObservANT will always act as the slave. The debug channel should be configured as bidirectional to allow for reverse direction debug messages (see section 3.2). The device being tested may also be connected to any number of other devices as normal. There either may be a separate debug channel on which the device and ObservANT communicate, or the device may interleave the debug messages within its regular messages. In the latter case, ObservANT will ignore non-debug messages. The figure below shows an example configuration where the device under test has 2 other nodes it is connected to as part of its normal operation and has one bidirectional debug channel.

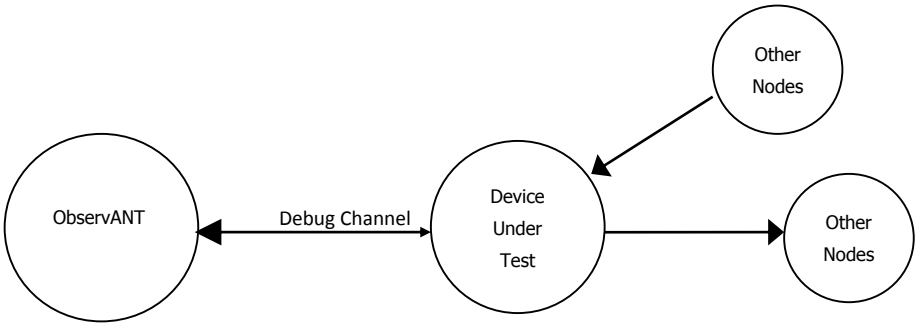


Figure 2-1. Example Topology

ObservANT may connect to many nodes at the same time using several debug channels. This allows for the monitoring of several different parts of a network simultaneously.

3 Debug Messages

Debug messages must have their first byte set to 0xF9 to indicate that it is indeed a debug message. Debug messages may be sent on a separate 'debug channel' or may be interleaved within the device's regular message pattern. ObservANT also may send debug messages in the reverse direction to instruct the device to change the information being sent.

3.1 Forward Direction Debug Messages

Forward direction debug messages are sent as broadcast messages from the device under test to ObservANT. Each debug message holds 2 'debug fields' and a 'fast debug byte'.

Table 2-1. Forward-direction debug message structure

Byte	Description	Length	Value	Rollover or Valid Range
0	Debug Message Indicator	1 byte	0xF9	N/A
1	Fast Debug Byte	1 byte	Byte which will be transmitted with each debug message.	0-255
2	1st Debug Field Index	1 byte	The index of the first debug field contained within the message. A value of 0xFF (255) indicates the first debug field is unused.	0-255
3	1st Debug Field Value LSB	2 bytes	The value of the first debug field contained within the debug message	0-65535
4	1st Debug Field Value MSB			
5	2nd Debug Field Index	1 byte	The index of the second debug field contained within the message. A value of 0xFF (255) indicates the second debug field is unused.	0-255
6	2nd Debug Field Value LSB	2 bytes	The value of the second debug field contained within the debug message	0-65535
7	2nd Debug Field Value MSB			

3.1.1.1 Debug fields

Debug fields are composed of an index and a value. If more than two debug values are to be transmitted, a rotation of messages should be implemented on the device side so each field gets transmitted in a timely manner. For instance, if debug fields with indexes 1, 2, and 3 are to be transmitted, the device could transmit in the pattern:

Message 1: Indexes 1 and 2

Message 2: Indexes 3 and 1

Message 3: Indexes 2 and 3

and so on. This is just a guideline and may be changed as the developer sees fit. For example, a particularly urgent debug fields might be sent with every debug message with the free field being used to cycle through the other indexes.

3.1.1.2 Fast Debug Byte

The fast debug byte is a single byte which will be transmitted with every debug message. This might be useful if the developer is waiting for a time sensitive signal from the device and cannot wait for the debug message rotation. For example, the developer might want to know immediately if a search has timed out on a particular channel. Using the fast debug byte ensures that the delay will be no greater than one debug channel period.

3.2 Reverse Direction Debug Messages (Filter Messages)

Filter messages are sent as acknowledged messages from ObservANT to the device being debugged (i.e. in the reverse direction). Filter messages can be used to select which debug fields out of a rotation are sent. Upon receipt of a 'Set Filter' message, the device being debugged should adjust its rotation of forward direction debug messages to include only fields with the indexes listed in the filter message. Multiple filter messages may be sent in succession to set more than 5 indexes. Upon receipt of a 'Reset Filter' message, the device should resume sending all available debug fields. The format of filter messages is described in the table below.

Table 2-2. Filter message structure

Byte	Description	Length	Value	Rollover or Valid Range
0	Debug Message Indicator	1 byte	0xF9	N/A
1	Filter Message Indicator	1 byte	0x03	N/A
2	Type of Filter Message	1 byte	0x01 – Set debug filter 0x02 – Reset debug filter	N/A
3	Filter Index 1	1 byte	Index of the debug field to be filtered 0xFF – unused	0-255
4	Filter Index 2	1 byte	Index of the Debug Field to be filtered 0xFF – unused	0-255
5	Filter Index 3	1 byte	Index of the Debug Field to be filtered 0xFF – unused	0-255
6	Filter Index 4	1 byte	Index of the Debug Field to be filtered 0xFF – unused	0-255
7	Filter Index 5	1 byte	Index of the Debug Field to be filtered 0xFF – unused	0-255

3.3 Reverse Direction Freeform Messages

It is possible to send a single freeform acknowledged message from ObservANT to device being debugged. How or if this feature is used is up to the developer. A possible use is to send a signal to the device being tested to cause it to change state from idle to a running state to begin testing.

4 Overview

4.1 Device Window

The first window that appears when launching ObservANT is the Device Window. The Device Window contains a list of Device Items which each represent an available ANT USB device plugged into the machine.

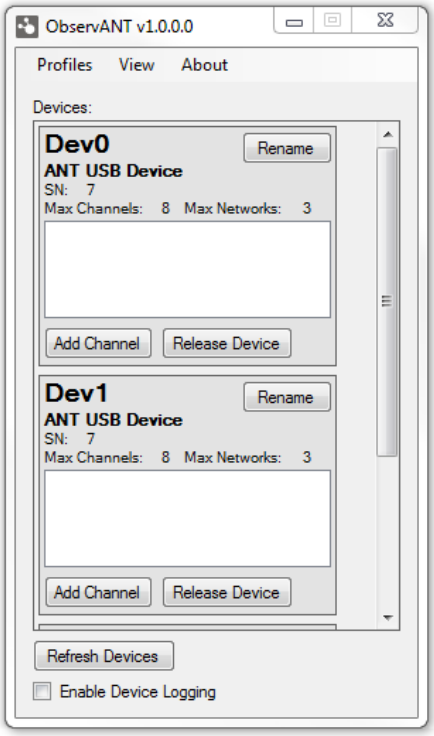


Figure 4-1. Device Window

4.1.1 Device Item

Each Device Item contains controls to interact with the device and information about that particular device such as serial number, maximum number of channels and maximum number of networks.

Channels may be added to the device by clicking the Add Channel button. The name of the device can be modified using the Rename button. Network keys can be set for each device by right clicking on the device item and selecting the Edit Networks option. Networks may only be configured before any channels have been added to the device.

Clicking Release Device will remove any open channels and release the device so it can be used by another application.

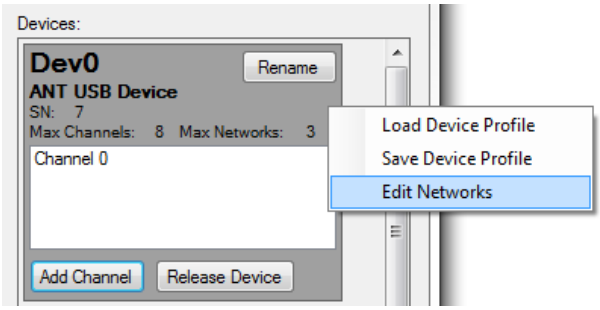


Figure 4-2. Device Item

Clicking the Add Channel button in a Device Item will add a channel to its list of channels and a new Channel Window will be launched.

4.2 Channel Window

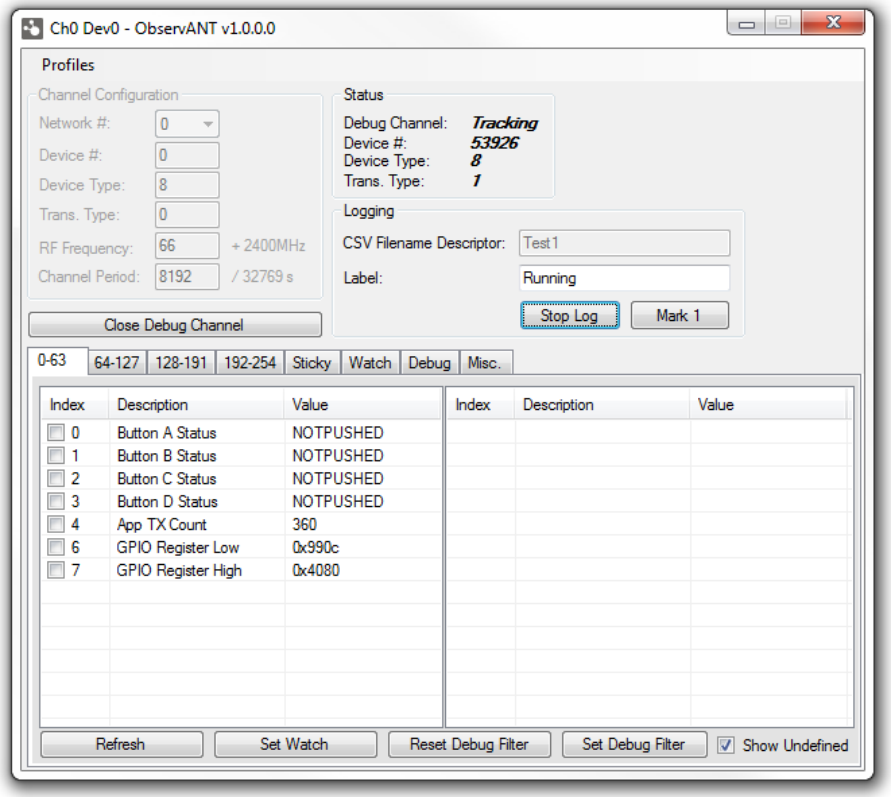


Figure 4-3. Channel Window

Channel parameters can be set up in the Channel Configuration box. Network 0 is set to the Public Network by default (but networks may be edited by right clicking on a Device Item). Device Number, Device Type and Transmission Type may be wildcarded by setting their value to 0. Once the channel parameters have been filled in, pressing the Open Channel button will assign and open the channel.

The Status box will display the parameters of the channel which is currently being tracked.

The Logging box contains various logging controls. See Section 6 for more information.

4.2.1 Debug Field Tabs

The first 4 tabs in the Channel Window contain the Debug Fields pairs. Each time a debug message containing a specific index is received, the corresponding row will be updated with the new value.

The description and the formatting of the value can be changed using the config.txt (see section 5). By default, the 2 byte debug values are shown as both an unsigned and signed numbers. For example, with no config.txt file specified, the value 0x8001 would be displayed as shown in Figure 4-4.

Debug filters (see section 3.2) can be set by checking the checkboxes next to the fields which should be filtered and clicking the Set Debug Filter button at the bottom of the window.

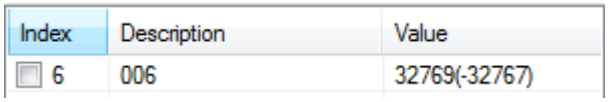


Figure 4-4. Default Debug Field Representation

4.2.2 Watch Tab

The watch tab behaves in the same way as the other debug field tabs. To watch a particular debug field, check the box next to it and click the Set Watch button at the bottom of the window. The Watch Tab is useful for grouping a set of important Debug Fields together on one page for easier viewing.

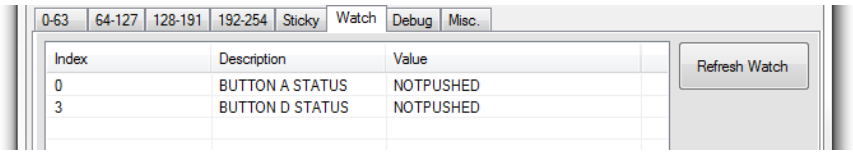


Figure 4-5. Watch Tab

4.2.3 Debug Tab

The debug tab displays the raw ANT messages received by the ObservANT. There is an option to view only debug messages (i.e. those whose first byte = 0xF9) or to view all messages on the channel. It is not recommended to use this feature for overnight logging as it may cause the program to run slowly.

4.2.4 Misc. Tab

The Misc. Tab contains reception statistics about the Debug Channel and the Fast Debug Byte. Freeform reverse direction acknowledged messages can also be sent from this tab.

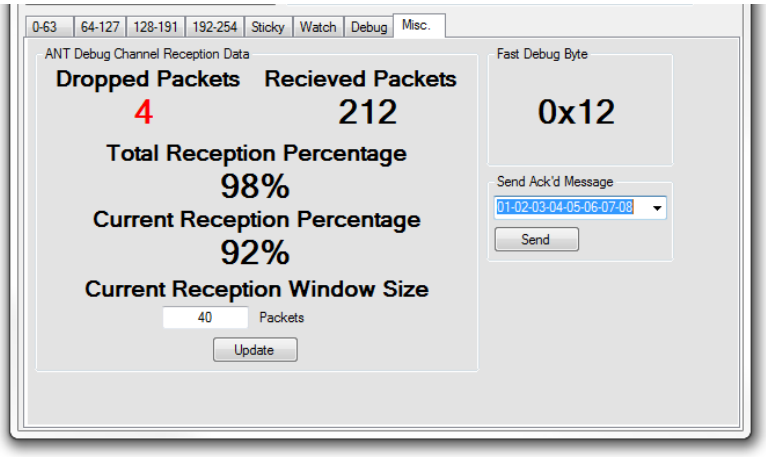


Figure 4-6. Misc. Tab

4.2.5 Sticky Tab

Indexes may be marked as Sticky through the config.txt file (see section 5). An index which is marked 'Sticky' will appear in the Sticky Tab. Every time the value changes, it is written under the previous value with a timestamp. This may be useful for watching the progression of states or for monitoring when a particular event took place.

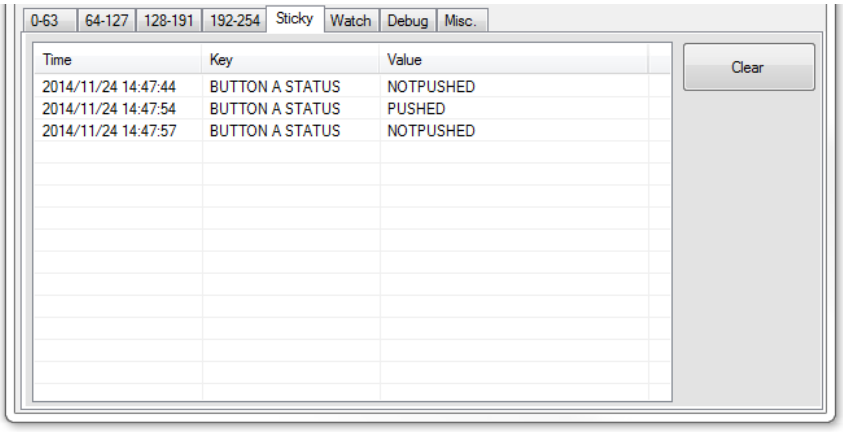


Figure 4-7. Sticky Tab

5 Configuration File

The config.txt file changes how the message content is displayed in the ObservANT. The configuration file must be called config.txt and be placed in the same directory as the executable.

5.1 Format

Lines in the Configuration File must take the following form:

Index,Description,Format,ScaleOrEnumerationList,Sticky

- Index: The index of the message in question
- Description: A text description of what the message is
- Format: How the message will be displayed. s,u,h,e,c or i corresponding to signed, unsigned, hex, enumeration, char or ignore respectively
- ScaleOrEnumList:
    - If the format is enumeration, a list of enumerations is put here. Each enumeration type is separated by a period (.)
    - If the format is a number type, an integer scale can be put here. The value will be divided by the scale before being displayed.
- Sticky: An 's' here indicates the value is 'sticky'. Sticky values will be displayed in the sticky list. Put any other character or leave blank to disable sticky

Lines beginning with the character '#' are considered comments and will be ignored.

5.2 Examples

Consider the following lines in a config.txt file:

- `00,sampleHexValue,h,1`

  The debug field with index 0 will have the description "sampleHexValue", display its value as an unscaled hex value.

- `01,sampleStickyEnumerationState,e,START.RUNNING.STOPPED,s`

  The debug field with index 1 will have its description set to "sampleStickyEnumerationState". Values of 0x000 will appear as START, 0x0001 will appear as RUNNING and 0x0002 will appear as STOPPED. Since there is an 's' as the last parameter, this value will show up in the Sticky Tab.

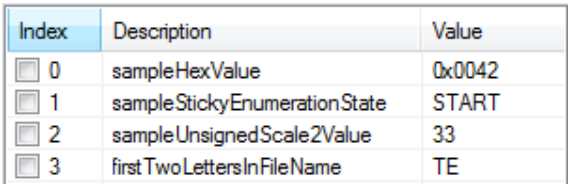
- `02,sampleUnsignedScale2Value,u,2`

  The debug field with index 2 will have the description "sampleUnsignedScale2Value" and display its value as an unsigned number which has been be scaled by 2.

- `03,firstTwoLettersInFileName,c`

  The debug field with index 3 will have the description "firstTwoLettersInFileName" and display its value as a pair of 8-bit ASCII characters. The first character displayed corresponds to the 1st Debug Field Value LSB, while the second character corresponds to the 2nd Debug Field Value MSB in the forward direction debug message structure.

Note: If the intention is to display as a character and make the field sticky a scale of 1 should be specified before the 's' for sticky.



Index	Description	Value
0	sampleHexValue	0x0042
1	sampleStickyEnumerationState	START
2	sampleUnsignedScale2Value	33
3	firstTwoLettersInFileName	TE

Figure 5-1. Example Configured Debug Messages

A sample config.txt file is included with ObservANT.

6 Logging

6.1 CSV Logs

ObservANT can automatically log a debug session in a CSV file on a channel per channel basis using the Logging Box in the Channel Window.

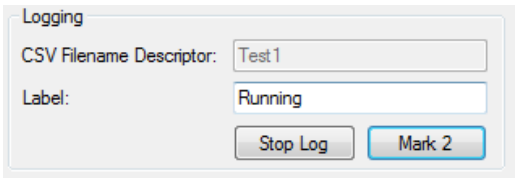


Figure 6-1. Example Configured Debug Messages

The CSV File Descriptor is a string which will be included in the filename for easier identification. Labels can be used to note when interesting events occurred so these points in time can be identified in the logs later. Pressing the Mark button will apply the label to all following messages, until the label is changed and Mark is pressed again. This is useful for marking a specific region of testing, for example, marking the time when a stimulus is applied to an ANT based sensor.

6.2 Device Logs

Device logging may be enabled from the Device Window. This logs all ANT messages for every device in the list and stores them as Device0.txt, Device1.txt, etc. This is the same type of log as is generated using other ANT PC tools including ANTwareII and SimulANT+.

7 Saving and Restoring Profiles

Device and channel settings can be saved and restored. ObservANT saves these settings in 'profiles' which are XML files detailing a particular set up of a channel, device or group of devices. The XML profile files may be opened in a text editor to view the saved values. Multi device profiles may contain many single device profiles, and each single device profile may contain many channel profiles.

ObservANT profiles have the same structure as the profiles created in ANTwareII and are mostly compatible.

Do not confuse these ObservANT profiles with ANT+ Device Profiles.

7.1 Channel Profiles

Channel profiles contain enough information to recreate the configuration of a Debug Channel. To save a profile, click Profiles->Save Channel Profile on the top bar of the channel window. To apply a saved channel profile, select Profile->Open Channel Profile from the top menu of the channel window.

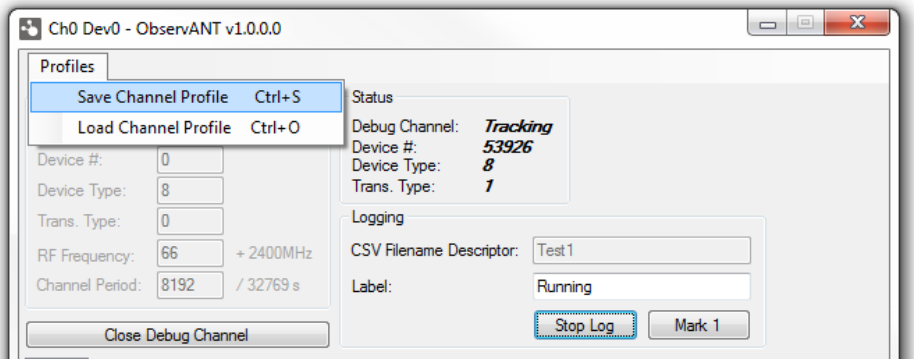


Figure 7-1. Saving a Channel Profile

7.1.1 Specifying Different Configuration Files for Specific Channels

The default config.txt file may be changed to a different file on a channel by channel basis by directly modifying the XML profile. Note that specifying a non-default configuration file will make the profile no longer compatible with ANTwareII. For example, adding the following line to a channel profile will cause this channel to use differentConfig.txt as its configuration file.

```
<configurationFilePath type="System.string">c:\differentConfig.txt</configurationFilePath>
```

7.2 Single Device Profiles

Device profiles contain enough information to recreate a single device item. They include information such as device name, configured channels and configured networks. A single device profile can be saved by selecting the device from the Device Window and then selecting Profiles->Save Single Device Profile from the top menu

Alternatively, right clicking on the device opens a context menu with a save option. Device profiles may be loaded by right clicking the device and selecting Load Device Profile from the context menu.

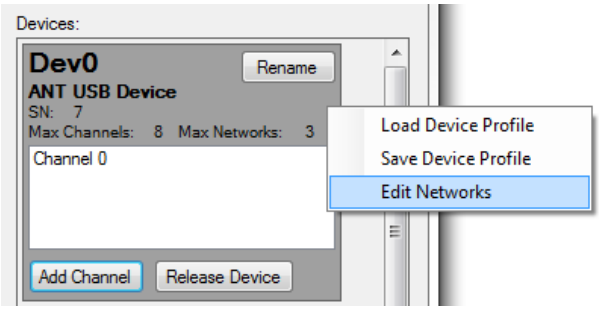


Figure 7-2. Saving and Loading Single Device Profile

7.3 Multi Device Profiles

Selecting Profiles -> Save Multi Device Profile from the top menu of the Device Window will save the profiles of all the listed devices. Loading a multi device profile is done by selecting Profiles -> Load Device Profile(s) from the top menu. When loading multi device profiles, ObservANT will attempt to fit the profile into whatever devices are currently in use. When loading a multi device profile, specific device profiles may not end up being reassigned to the exact device they were created on, so changing the names of the devices before saving the profile is advised.

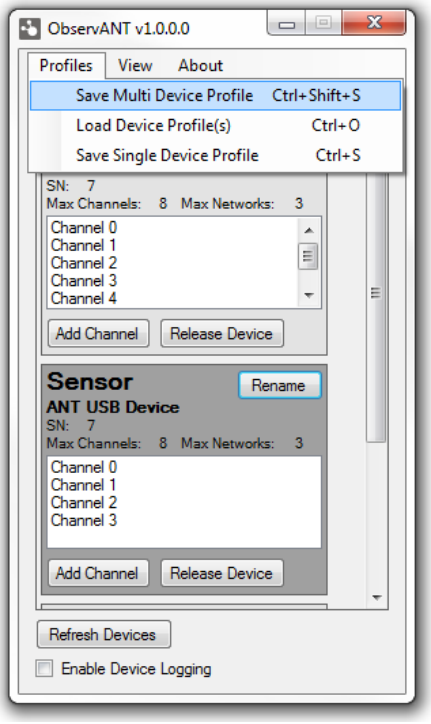


Figure 7-3. Saving and Loading a Multi Device Profile