AutoANT Scripts

ABSTRACT

The ability to transmit specific data in a controlled, repeatable manner enhances the accuracy and speed of ANT device development, testing, and diagnostics. AutoANT Scripts provide developers with this ability. This application note provides an overview of AutoANT Scripting, and describes how to create scripts manually as well as with a device simulator.

TABLE OF CONTENTS

LIST OF FIGURES

1 Introduction

AutoANT is a simplistic script language used to control sending and receiving ANT serial messages to and from an ANT device. It was created to allow automation of many testing and debugging scenarios, and then developed into a general purpose tool that can be used for a wide variety of scenarios. This document describes the script language and how it can be used, including examples. To run the scripts, ANTWare II now includes an AutoANT script engine which can execute scripts written in the AutoANT format.

For more information on using the ANTware II AutoANT tool, refer to the ANTware II User Guide.

1.1 AutoANT Script Overview

The simplest way to use an AutoANT script is to control an ANT enabled USB device through the ANTware II interface. Scripts can be used to sequence pre-determined ANT serial messages, providing the user with the ability to automate (and hence easily reproduce) their desired interactions with ANT.

AutoANT scripts can also be used in other ways, from simple time saving configuration scripts to more complicated scripts for advanced configuration procedures, repetitive or multi-part data messaging sequences, device simulations, or to recreate transmissions recorded from an ANT debug log file.

Used correctly, AutoANT scripts are powerful tools for ANT development; however, the user should be aware that AutoANT is not a fully-featured scripting language. Most notably, AutoANT does not support conditional branching. If greater flexibility than AutoANT can provide is required, then developing a custom application using the ANT Library Package is the recommended approach. The ANT Library Package can be found at www.thisisant.com.

2 Relevant Documents

It is strongly recommended that the following documents be reviewed prior to using this application note:

- ANT Message Protocol and Usage
- ANTware II User Guide
- Relevant ANT+ Device Profile(s)

3 Script Elements

The AutoANT script language utilizes only a simple set of operations:

- Write (`w`) – sends a scripted ANT serial message to the ANT device
- Read (`r`) – waits to receive the scripted ANT serial messages from the ANT device
- Pause (`p`) – pauses the script engine for a defined length of time
- `PauseBreak` – pauses the script engine until the user manually resumes the script
- `Loop` – allows the contents of a loop to be repeated a finite number of times

Descriptions of AutoANT language features, operations, and examples of their usage are provided below. Examples in this section are excerpts from the example script found in section 4.

3.1 Version Tag

A version tag is found at the beginning of all AutoANT scripts:

```
###ANT_SCRIPT_VERSION: 0.01
```

It is important to recognize that the version tag is present to define the minimum version of the script engine that is required, not to denote the version number of the script itself. It is in place to ensure forward compatibility with future versions of the script engine.

3.2 Script Code Comments

In any .ants file, comments are denoted using the '#' character. Anything typed after a '#' character will be ignored by the script reader. Comments will be included in the accompanying figures in this document to highlight important information.

3.3 Printing to Output

Text can be output to the script output field of the script engine (typically the screen) by using the 'o' command:

```
o*****************************************************
o This text will be displayed in the script output field
o*****************************************************
```

3.4 Commands

To write (send) and read (receive) an ANT serial message in a script, begin with the message ID followed by the data payload. Do not include the sync byte, message length or checksum fields. Each byte is represented using hexadecimal values, delimited within square brackets (e.g. `0x7B` is written as `[7B]`). For more information regarding the format of ANT messages, refer to the ANT Message Usage and Protocol document.

Note that whitespace is ignored in AutoANT commands, hex values do not need to be zero padded, and that the AutoANT script reader is NOT case sensitive:

- `[A]` is the same as `[0A]`
- `[3][0 ][2  d]    [c]` is the same as `[03][00][2D][0C]`

3.4.1 Write Commands

'Write' commands contain the ANT serial message to be sent to the ANT device, either to configure the chip or to transmit data over the wireless channel. The `'w'` character indicates that the serial message is a write command:

```
w [42][00][10][00] # Assign ch0, bidir master, netwk0
```

3.4.2 Read Commands

The `'r'` command is used to indicate that the script should wait for a specific serial response message before continuing.

A read command specifies the exact format of the serial response message that the application shall wait for. The command will look for a message which matches both the content and number of bytes given in the call.

An example of an `'r'` command:

```
w [42][00][10][00] # Assign ch0, bidir master, netwk0
r [40][00][42][00] # Wait for RESPONSE_NO_ERROR (default timeout = 3s)
```

In this case, the script is waiting for a RESPONSE_NO_ERROR reply to the assign channel command.

3.4.2.1 Response Buffer

In order to monitor responses reliably when searching for matches for read commands, the script engine

buffers incoming messages, rather than attempting to monitor them in real time. This buffer is cleared whenever a write operation occurs; a read command completes successfully; or when the script resumes execution from a pause or other stoppage.

This means whenever a read operation occurs, all messages since the last buffer clearing event are searched; not necessarily those only occurring after the time of its call. Note that an unsuccessful read call does NOT clear the response buffer.

3.4.2.2 Response Timeout

All read commands shall have an associated timeout parameter, which determines how long to wait for the response before reporting failure and then continuing execution of the script. This timeout value can be specified manually by including the number of milliseconds to wait as an integer, or use the default timeout by not specifying a value, as shown below. The default timeout is initialized to 3000 on start-up, but can be manually changed for any subsequent operations in the script by calling the 'r' command with a time parameter and no serial message specified:

```
w [42][00][10][00] # Assign ch0, bidir master, netwk0
r [40][00][42][00] # Wait for RESPONSE_NO_ERROR (default timeout = 3s)


w [51][00][E9][03][2B][05]     #Set Channel ID. Dev# 1001, DevType 43, TxType 5
r100 [40][00][51][00]          #Wait for RESPONSE_NO_ERROR (100ms response timeout)
r5000                          #Set default timeout to 5s
```

If an 'r' command exceeds the specified timeout value, a failure will be displayed in the script output field but the program will continue running the script.

3.4.2.3 Critical Response Flag

Read commands are often essential for the script to continue in a known state. In these cases, the 'critical' response flag is available which will cause the script to display an error and stop processing when the read command times out, instead of continuing execution. A read command is flagged as a critical response by including the '!' character immediately following an 'r':

```
w  [43][00][00][20]    #Set Channel Period: (8192/32768) = 4Hz
r! [40][00][43][00]    #Wait for RESPONSE_NO_ERROR (Critical response 5s timeout)
w  [45][00][42]        #Set Channel RF Frequency
r!300 [40][00][45][00] #Wait for RESPONSE_NO_ERROR (Critical response 300ms timeout)
```

3.4.2.4 Wildcard Values

A wildcard value of '?' may be used in message definitions to allow more flexible matching than an exact match. The wildcard value on its own denotes matching of any value for that entire byte (ie: "don't care"). If the wildcard value is indicated with a specific byte value, the response is matched bitwise against only the high bits of the specified value. For Example:

- `[40][00][01][?]` indicates the application shall wait for any RF channel event on channel 0.
- `[40][00][01][?B0]` this line will pass if the last byte of the incoming response matches 1x11xxxx (0xB0) where x means "don't care".

Some examples of valid read commands using wildcards:

```
w [4E][00][84][FF][51][21][A6][24][0B][48]    #Broadcast random data packet on ch0
r [40][00][01][?]                             #Wait for any RF Event

w [4D][00][54]                                #Request Capabilities from ANT
r! [54][?][?][?][?09][?][?]                   #Check to make sure that the device has the
                                              #CAPABILITIES_SEARCH_LIST_ENABLED
                                              #and CAPABILITIES_NETWORK_ENABLED bits set
                                              #high in its advanced options
                                              #(see pg 89 in Protocol and Usage document)
```

3.4.3 Pause Commands

Pause commands pause execution of the script for a specified period of time. They are called using the 'p' character followed by an integer number representing the time to wait (or pause) in milliseconds. The response buffer is cleared after each 'p' command when the script is resumed. This means that immediately after a 'p' call, an 'r' call will fail as no data will be available in the buffer for the 'r' command to compare. Because of this behavior, the 'p' command can also be used to manually clear the response buffer:

```
w [4E][00][84][FF][51][21][A6][24][0B][48]
p1000                                         # Pause script engine for 1s
                                              #(ANT will Tx above data for 1s)

w [4E][00][23][FA][B4][15][06][88][00][CC]    #Broadcast another random data packet on
ch0
r [4E][ ?][ ?][ ?][ ?][ ?][ ?][ ?][ ?][ ?]    #Wait for any broadcast packet from the
                                              #slave.
p0                                            #clears response buffer
```

In this example, the read command indicates that the script wait for a broadcast data message. If no data is received, the 'r' command will timeout after 5s. During this time, the response buffer may have accumulated multiple serial messages that will not be cleared when the 'r' command times out. As such, the script should clear the response buffer to prevent future 'r' calls from reading this data. 'p0' on the final line clears the response buffer quickly so that future 'r' commands will behave reliably.

3.4.4 PauseBreak Commands

The AutoANT script engine can be forced to halt processing by using the 'pauseBreak' command. The behavior of a pauseBreak is identical to that of a user manually pressing the 'pause' button. This allows a script to be used in scenarios that require user input, or that require another script/application to execute. To resume from a pauseBreak command, use the Resume button in the ANTware AutoANT tool.

```
oPress Resume to Continue
pauseBreak                                    # Pause script engine until user resumes
```

3.4.5 Loop and Loopend Commands

The 'loop' and 'loopend' commands are used to implement loop functionality. The 'loop' command indicates the start of the loop and is followed by the desired number of iterations; the loopend command indicates the point at which the script engine should loop back to the beginning. This example implements nested loops.

```
loop(3)
     loop(10)
                 w [4E][00][84][FF][51][21][A6][24][0B][48]
          r [40][00][01][03]
     loopend

     oPress Resume to Continue
     pauseBreak
loopend
```

4 Example Script

The following script can be copied and pasted into an autoANT .ants file, and will work with ANTware II. This script properly implements elements from Section 3, however it is not meant to perform any specific task other than to transmit meaningless data according to the scheduling in the script.

```
###ANT_SCRIPT_VERSION: 0.01


o*****************************************************
oThis text will be displayed in the script output field
o*****************************************************



#############WRITE AND READ EXAMPLE############################
w [42][00][10][00]            # Assign ch0, bidir master, netwk0
r [40][00][42][00]            # Wait for RESPONSE_NO_ERROR (default timeout = 3s)

w [51][00][E9][03][2B][05]    #Set Channel ID. Dev# 1001, DevType 43, TxType 5
r100 [40][00][51][00]         #Wait for RESPONSE_NO_ERROR (100ms response timeout)
r5000                         #Set default timeout to 5s



#############CRITICAL RESPONSE EXAMPLE############################
w  [43][00][00][20]           #Set Channel Period: (8192/32768) = 4Hz
r! [40][00][43][00]           #Wait for RESPONSE_NO_ERROR (Critical response 5s timeout)
w  [45][00][42]               #Set Channel RF Frequency
r!300 [40][00][45][00]        #Wait for RESPONSE_NO_ERROR
                              #(Critical response 300ms timeout)

w [4B][00]                    #Open Channel 0
r [40][00][4B][00]            #Wait for RESPONSE_NO_ERROR (response timeout = 5s)



#############WILDCARD EXAMPLE#####################################
w [4E][00][84][FF][51][21][A6][24][0B][48]    #Broadcast random data packet on ch0
r [40][00][01][?]                             #Wait for any RF Event

w [4D][00][54]                                #Request Capabilities from ANT
r! [54][?][?][?][?09][?][?]                   #Check to make sure that the device has the
                                              #CAPABILITIES_SEARCH_LIST_ENABLED
                                              #and CAPABILITIES_NETWORK_ENABLED bits set
                                              #high in its advanced options
                                              #(see pg 89 in Protocol and Usage document)
```

```
#############PAUSE EXAMPLE########################################
w [4E][00][84][FF][51][21][A6][24][0B][48]
p1000                                          # Pause script engine for 1s
                                               #(ANT will Tx above data for 1s)

w [4E][00][23][FA][B4][15][06][88][00][CC]     #Broadcast another random data packet on
ch0
r [4E][ ?][ ?][ ?][ ?][ ?][ ?][ ?][ ?][ ?]     #Wait for any broadcast packet from the
                                               #slave.
p0                                             #clears response buffer



#############PAUSEBREAK EXAMPLE###################################
oPress Resume to Continue
pauseBreak                                     # Pause script engine until user resumes



#############LOOP EXAMPLE#########################################
loop(3)
     loop(10)
                 w [4E][00][84][FF][51][21][A6][24][0B][48]
          r [40][00][01][03]
     loopend

     oPress Resume to Continue
     pauseBreak
loopend
```

5 Producing a Script

5.1 Manual Script Generation

To manually create a script, the user must be familiar with ANT message protocol and usage, as well as the desired ANT serial message sequence to reproduce. The user may use the AutoANT scripting elements discussed in section 3 to create custom scripts in any plain text editor.

When manually writing AutoANT scripts, the user must consider which fields of the ANT serial message to include. Figure 5-1 illustrates the ANT serial message format. Refer to the ANT Message Protocol and Usage document for a full description of all ANT serial messages and their respective formats. AutoANT scripting only requires the Message ID and Message Content bytes as outlined in green. The SYNC, Message Length and Check sum fields are NOT required.

Figure 5-1. AutoANT Script Message Bytes

When manually creating scripts, the user must consider the presence and timing of any associated response messages. For example, if the user requires that the script continue regardless of responses, omitting ‘r’ commands may be an appropriate solution. However, if the user requires the script to interact with the ANT chip and be aware of any response messages, the read commands should be used.

5.2 Automatic Script Generation

Scripts can also be generated automatically using log files from any ANT PC application with logging capabilities, such as ANTware II or SimulANT+. When logging is enabled, the ANT PC application creates a text log file named DeviceX.txt, where 'X' is the USB port number used for the connected ANT device. The default location of the log file depends on the application; please refer to the corresponding user manual for details.

To generate the AutoANT script, the log can be processed using the AutoANT device log parsing tool. This tool is a perl script that comes bundled with this application note. To download this tool again, visit http://www.thisisant.com/developer/resources/downloads/ and choose AN17 AutoANT Scripts. Perl must be installed to run the script, and is available through many online resources.

The following example uses a SimulANT+ log file to automatically generate an AutoANT script.

Steps:

1. Open and configure SimulANT+ to transmit the desired data. A device log will be generated in the working directory of the application. Figure 5-2 shows an example of the SimulANT+ having been used to configure and open a channel, transmit HRM data, and close the channel.

   Figure 5-3 shows an example log generated while simulating a heart rate sensor. Comments and whitespace have been added to help clarify the output. Note that the network key is blanked. Do not modify the device log, other than to replace the blank network key with the ANT+ network key. Modifying the device log may lead to unpredictable results.

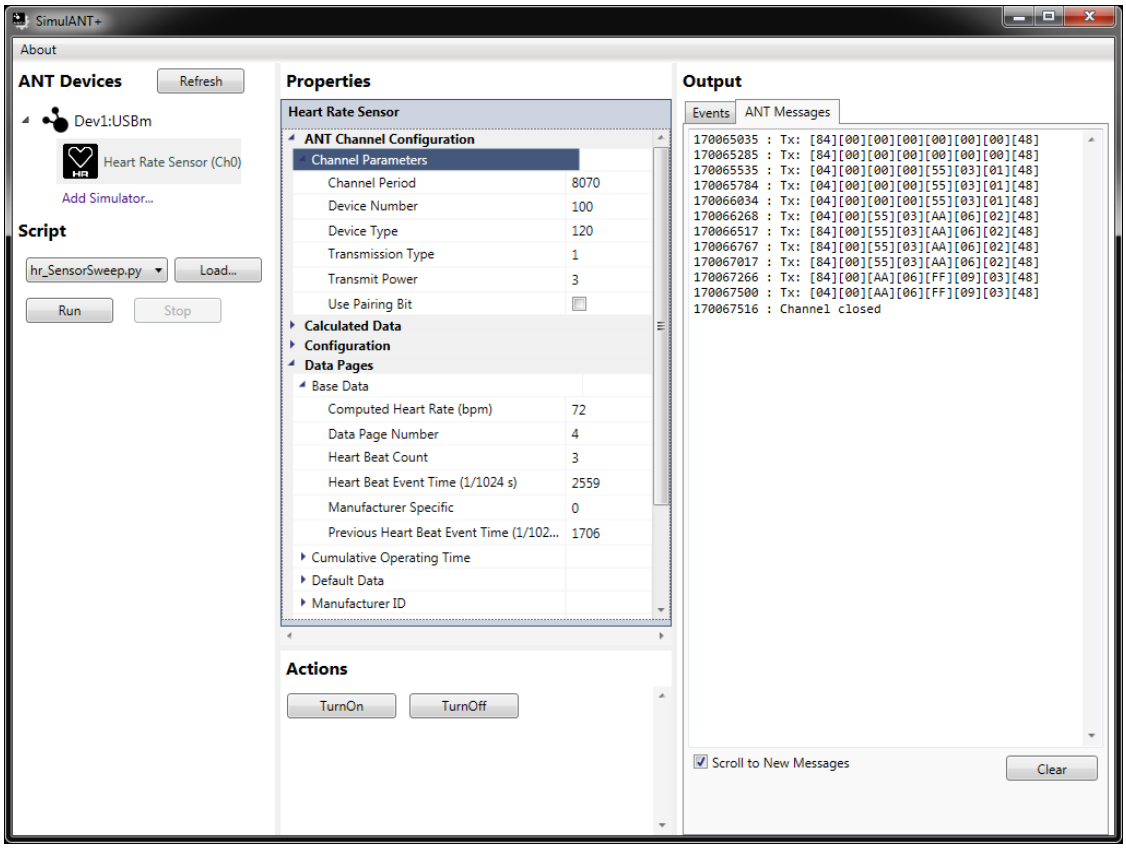
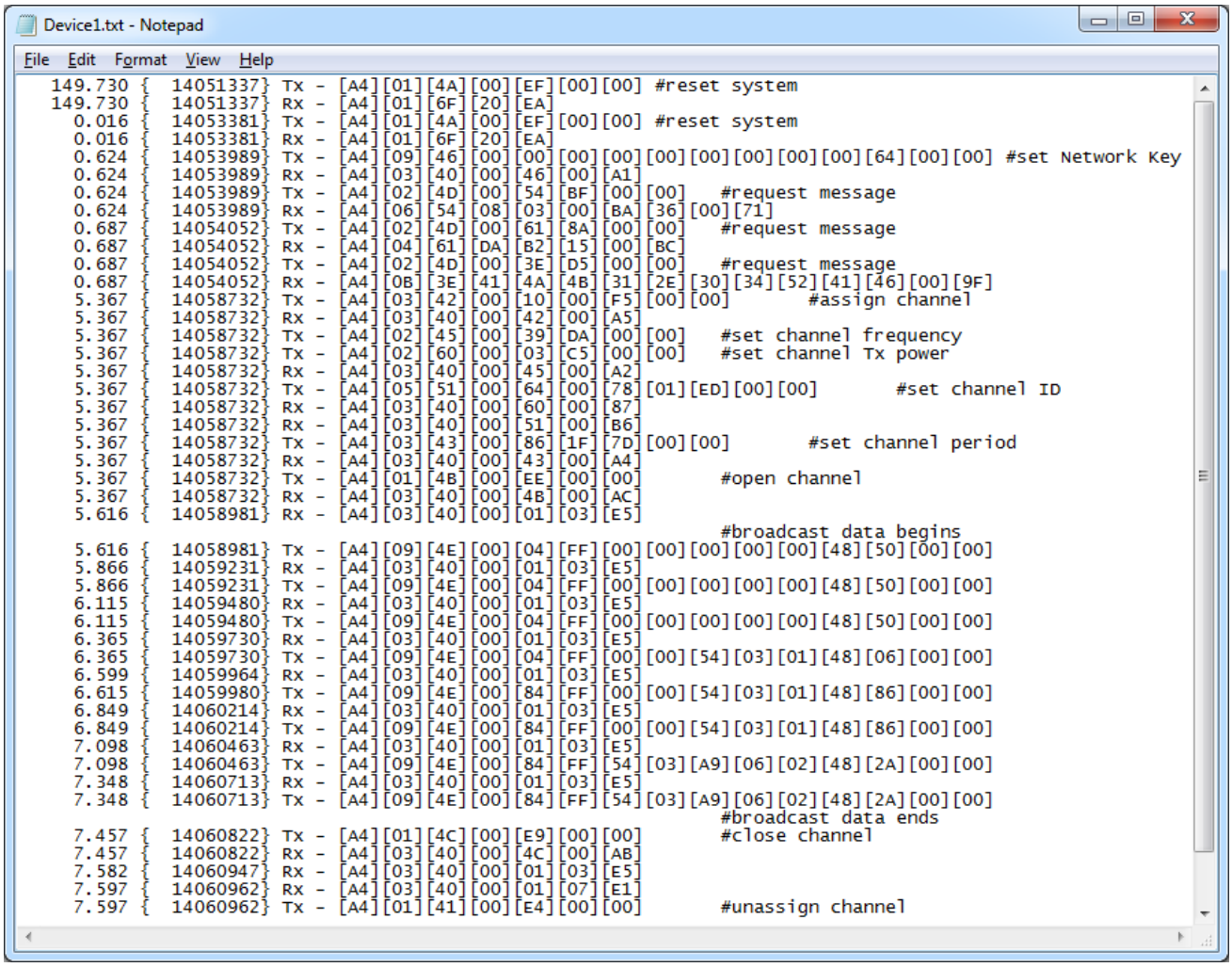


Figure 5-2. SimulANT+



Device1.txt - Notepad

File Edit Format View Help

```
  149.730 {  14051337} Tx - [A4][01][4A][00][EF][00][00] #reset system
  149.730 {  14051337} Rx - [A4][01][6F][20][EA]
    0.016 {  14053381} Tx - [A4][01][4A][00][EF][00][00] #reset system
    0.016 {  14053381} Rx - [A4][01][6F][20][EA]
    0.624 {  14053989} Tx - [A4][09][46][00][00][00][00][00][00][00][00][64][00][00] #set Network Key
    0.624 {  14053989} Rx - [A4][03][40][00][46][00][A1]
    0.624 {  14053989} Tx - [A4][02][4D][00][54][BF][00][00]   #request message
    0.624 {  14053989} Rx - [A4][06][54][08][03][00][BA][36][00][71]
    0.687 {  14054052} Tx - [A4][02][4D][00][61][8A][00][00]   #request message
    0.687 {  14054052} Rx - [A4][04][61][DA][B2][15][00][BC]
    0.687 {  14054052} Tx - [A4][02][4D][00][3E][D5][00][00]   #request message
    0.687 {  14054052} Rx - [A4][0B][3E][41][4A][4B][31][2E][30][34][52][41][46][00][9F]
    5.367 {  14058732} Tx - [A4][03][42][00][10][00][F5][00][00]       #assign channel
    5.367 {  14058732} Rx - [A4][03][40][00][42][00][A5]
    5.367 {  14058732} Tx - [A4][02][45][00][39][DA][00][00]   #set channel frequency
    5.367 {  14058732} Tx - [A4][02][60][00][03][C5][00][00]   #set channel Tx power
    5.367 {  14058732} Rx - [A4][03][40][00][45][00][A2]
    5.367 {  14058732} Tx - [A4][05][51][00][64][00][78][01][ED][00][00]       #set channel ID
    5.367 {  14058732} Rx - [A4][03][40][00][60][00][87]
    5.367 {  14058732} Rx - [A4][03][40][00][51][00][B6]
    5.367 {  14058732} Tx - [A4][03][43][00][86][1F][7D][00][00]       #set channel period
    5.367 {  14058732} Rx - [A4][03][40][00][43][00][A4]
    5.367 {  14058732} Tx - [A4][01][4B][00][EE][00][00]       #open channel
    5.367 {  14058732} Rx - [A4][03][40][00][4B][00][AC]
    5.616 {  14058981} Rx - [A4][03][40][00][01][03][E5]
                                                       #broadcast data begins
    5.616 {  14058981} Tx - [A4][09][4E][00][04][FF][00][00][00][00][00][48][50][00][00]
    5.866 {  14059231} Rx - [A4][03][40][00][01][03][E5]
    5.866 {  14059231} Tx - [A4][09][4E][00][04][FF][00][00][00][00][00][48][50][00][00]
    6.115 {  14059480} Rx - [A4][03][40][00][01][03][E5]
    6.115 {  14059480} Tx - [A4][09][4E][00][04][FF][00][00][00][00][00][48][50][00][00]
    6.365 {  14059730} Rx - [A4][03][40][00][01][03][E5]
    6.365 {  14059730} Tx - [A4][09][4E][00][04][FF][00][00][54][03][01][48][06][00][00]
    6.599 {  14059964} Rx - [A4][03][40][00][01][03][E5]
    6.615 {  14059980} Tx - [A4][09][4E][00][84][FF][00][00][54][03][01][48][86][00][00]
    6.849 {  14060214} Rx - [A4][03][40][00][01][03][E5]
    6.849 {  14060214} Tx - [A4][09][4E][00][84][FF][00][00][54][03][01][48][86][00][00]
    7.098 {  14060463} Rx - [A4][03][40][00][01][03][E5]
    7.098 {  14060463} Tx - [A4][09][4E][00][84][FF][54][03][A9][06][02][48][2A][00][00]
    7.348 {  14060713} Rx - [A4][03][40][00][01][03][E5]
    7.348 {  14060713} Tx - [A4][09][4E][00][84][FF][54][03][A9][06][02][48][2A][00][00]
                                                       #broadcast data ends
    7.457 {  14060822} Tx - [A4][01][4C][00][E9][00][00]       #close channel
    7.457 {  14060822} Rx - [A4][03][40][00][4C][00][AB]
    7.582 {  14060947} Rx - [A4][03][40][00][01][03][E5]
    7.597 {  14060962} Rx - [A4][03][40][00][01][07][E1]
    7.597 {  14060962} Tx - [A4][01][41][00][E4][00][00]       #unassign channel
```

Figure 5-3. Simulator Device Log

2. Move the log file into the same folder as the parsing tool. It is helpful to rename the log file to something meaningful, however for simplicity the files have not been renamed in this example.

3. Use the parsing tool to extract the body of data from the text file, and produce an .ants file. This is done by:

    a. Opening a command prompt

    b. Navigating to the folder containing the parser and log file

    c. Using the following command:

        perl parse_AutoANT.pl [logFileName].txt >[outputFileName].ants

The '>' character tells the parsing tool to output to a file. Using '>>' will append the output to the designated file. The resulting .ants file will not contain the commands to configure, open, or close a channel. Configuration commands are omitted in this example script as the intent is to reproduce over the air data only. As such, this script should be executed over a preconfigured channel that is already open in ANTware II or other applicable program.

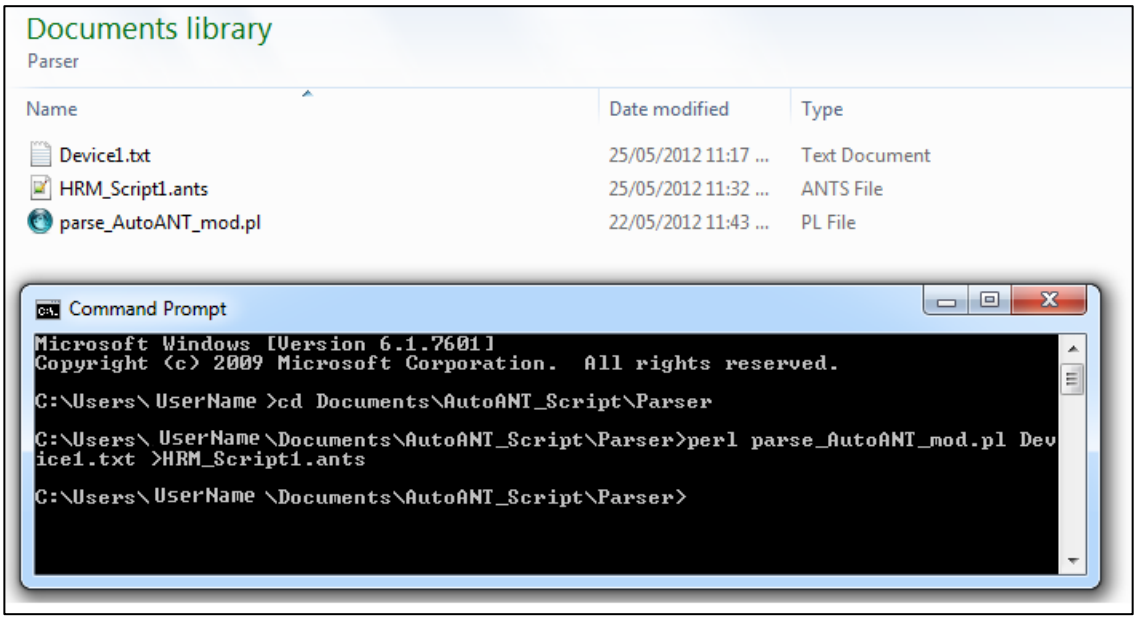


Figure 5-4. Using the Parsing Tool

4. Add the version tag to the top of the file. Any additional script information including messages, comments, or text output may now be edited manually.

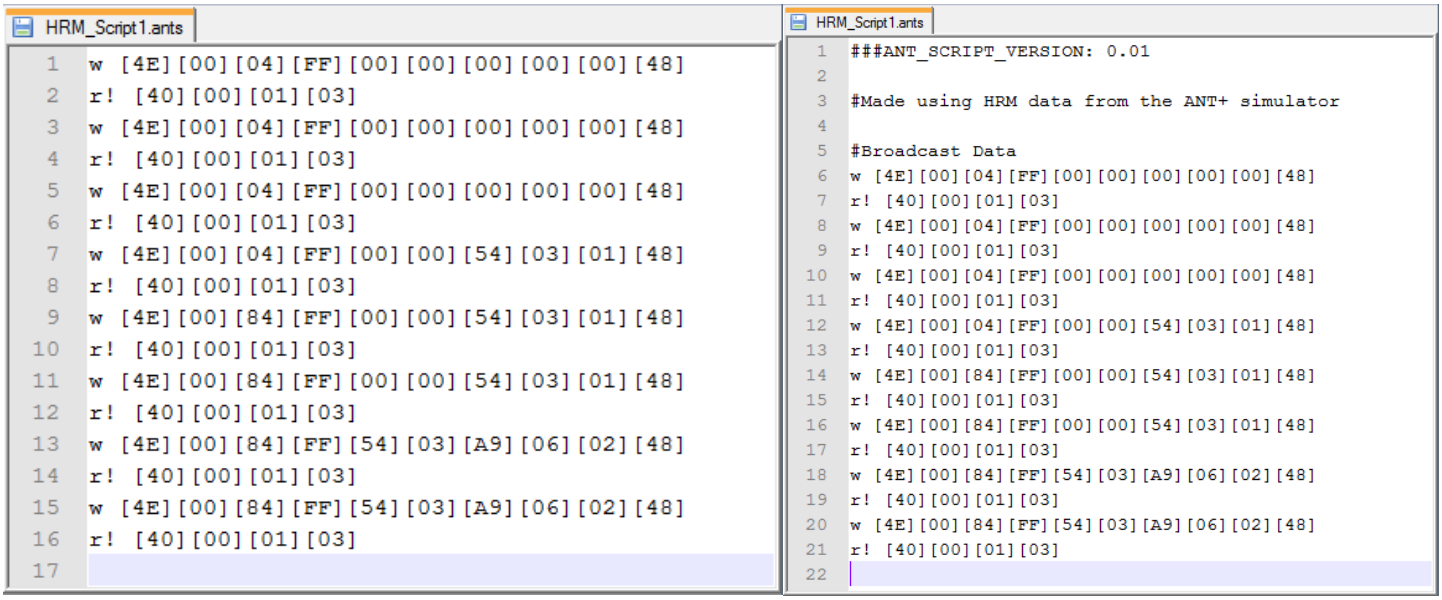


Figure 5-5. Parser Output and Completed Script

6 Closing Remarks

This application note describes AutoANT Script elements, formatting and generation. If any of the concepts presented in this application note are unclear or for any further inquiries, please go to www.thisisant.com