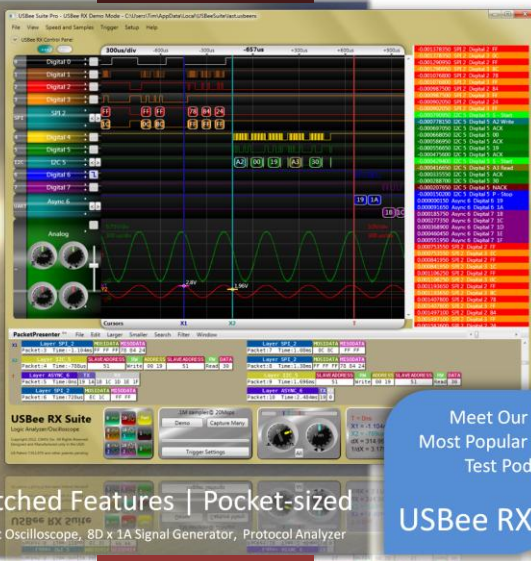


***Illuminating*** [ih-loo-muh-ney-ting] v.  
*Quickly expose the deepest details of your  
design to speed validation, test and debug*



**Fast Sampling | Huge Buffer | Unmatched Features | Pocket-sized**

Up to 600 Msps, 16 M Samples/Edges,  $\pm 60V$ , 18D x 2A Mixed Signal 10-bit Oscilloscope, 8D x 1A Signal Generator, Protocol Analyzer

Meet Our  
Most Popular  
Test Pod

**USBee RX**

# USBEE RX USERS MANUAL

CWAV, Inc.

[www.usbee.com](http://www.usbee.com)



# USBEE RX

## USERS MANUAL

CWAV, Inc.  
[www.usbee.com](http://www.usbee.com)  
(951) 694-6808  
[support@usbee.com](mailto:support@usbee.com)

## USBee RX Suite License Agreement

The following License Agreement is a legal agreement between you (either an individual or entity), the end user, and CWAV, Inc. makers of the USBee Test Pods and USBee RX Suite software. You have received or downloaded the USBee RX Suite Package ("Software"), which consists of the USBee RX Suite Software and Documentation. If you do not agree to the terms of the agreement, do not install and run this software.

By installing this USBee RX Suite software, you agree to be bound by the terms of this Agreement.

### Grant of License

CWAV provides royalty-free Software, both in the USBee Package and on-line at [www.usbee.com](http://www.usbee.com), for use with the USBee Test Pods and grants you license to use this Software under the following conditions: a) You may use the USBee Software only in conjunction with a USBee Test Pod, or in demonstration mode with no USBee Pod connected, b) You may not use this Software in conjunction with any pod providing similar functionality made by other than CWAV, Inc, and c) You may not sell, rent, transfer or lease the Software to another party.

### Copyright

No part of the USBee Package (including but not limited to Software, manuals, labels, USBee Pod, or accompanying diskettes) may be reproduced, stored in a retrieval system, or transcribed, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of CWAV, Inc, with the sole exception of making backup copies of the Software for restoration purposes on your own machine. You may not reverse engineer, decompile, disassemble, merge or alter the USBee Software or USBee Pod in any way.

### Limited Warranty

The USBee Software and related contents are provided "as is" without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, with the sole exception of manufacturing failures in the USBee Pod or diskettes. CWAV warrants the USBee Pod and physical diskettes to be free from defects in materials and workmanship for a period of 12 (twelve) months from the purchase date. If during this period a defect in the above should occur, the defective item may be returned to the place of purchase for a replacement. After this period a nominal fee will be charged for replacement parts. You may, however, return the entire USBee Package within 30 days from the date of purchase for any reason for a full refund as long as the contents are in the same condition as when shipped to you. Damaged or incomplete USBee Packages will not be refunded.

### Software Updates

The information in the Software and Documentation is subject to change without notice and, except for the warranty, does not represent a commitment on the part of CWAV. CWAV cannot be held liable for any mistakes in these items and reserves the right to make changes to the product in order to make improvements at any time. The Software will change from time to time to add new functionality, fix bugs and disable the use of non-CWAV made products that attempt to use this Software.

### Liability

IN NO EVENT WILL CWAV BE LIABLE TO YOU FOR DAMAGES, DIRECT, INDIRECT, INCIDENTAL OR CONSEQUENTIAL, INCLUDING DAMAGES FOR ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF THE USE OR INABILITY TO USE SUCH USBEE POD, SOFTWARE AND DOCUMENTATION, EVEN IF CWAV HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR FOR ANY CLAIM BY ANY OTHER PARTY. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU. IN NO EVENT WILL CWAV'S LIABILITY FOR DAMAGES TO YOU OR ANY OTHER PERSON EVER EXCEED THE AMOUNT OF THE PURCHASE PRICE PAID BY YOU TO CWAV TO ACQUIRE THE USBEE, REGARDLESS OF THE FORM OF THE CLAIM.

### Term

This license agreement is effective until terminated. You may terminate it at any time by returning the USBee Package (together with the USBee Pod, Software and Documentation) to CWAV. It will also terminate upon conditions set forth elsewhere in this agreement or if you fail to comply with any term or condition of this agreement. You agree that upon such termination you will return the USBee Package, together with the USBee Pod, Software and Documentation, to CWAV.

USBee RX User's Manual, Version 1.5

Copyright 2012 CWAV. All Rights Reserved



## CONTENTS

<b>INTRODUCING THE USBEE RX POD .....</b>	<b>15</b>
THE USBEE RX TEST POD.....	17
PC SYSTEM REQUIREMENTS.....	19
EACH PACKAGE INCLUDES.....	19
<b>INTRODUCING THE USBEE RX SUITE .....</b>	<b>21</b>
USBEE RX SUITE OVERVIEW .....	21
See the information you want FAST! .....	22
Data Acquisition over USB 2.0 .....	22
Superior Quality Design - Professional Results .....	22
Fast and Detailed Waveform Viewing .....	22
Measure It .....	23
Fully configurable Look and Feel .....	23
Serial Bus Analysis .....	24
Data Storage .....	26
PacketPresenter.....	27
Fast Pan Bus Viewing .....	28
Hot Keys.....	29
Smart Search.....	30
Sample and Smart Markers .....	30
Annotations and Sticky Notes.....	31
Acquisition Control .....	31
Display Modes .....	32
Analog Channels Scaling .....	32
USBee Suite Data File Importing.....	32
Browser-like Navigation.....	33
Complex Triggering.....	33
Relative Time Decode .....	34
USBEE RX CONTROL PANEL.....	35
<b>QUICK START .....</b>	<b>36</b>
PC SYSTEM REQUIREMENTS.....	36
SOFTWARE INSTALLATION .....	37
<i>Detailed Software Installation .....</i>	<i>38</i>
DEMO MODE.....	42
TESTING YOUR CIRCUIT USING THE USBEE RX .....	43
<b>USING THE USBEE RX SUITE MIXED SIGNAL OSCILLOSCOPE .....</b>	<b>44</b>
ANALYZER SETUP .....	44

<i>Quick Setup Configuration</i> .....	44
<i>Hot Keys</i> .....	48
<i>Signal Names</i> .....	48
<i>Buffer Sizes and Sample Rate settings</i> .....	48
<i>Setting Simple Triggers</i> .....	50
<i>Setting Complex Triggers</i> .....	52
CAPTURING WAVEFORM DATA.....	53
VIEWING CAPTURED DATA.....	53
<i>Scrolling, Zooming and Panning Waveforms</i> .....	53
<i>Modifying Wave lines</i> .....	55
<i>Decoding Bus Traffic Inline</i> .....	57
<i>Decoded Data List</i> .....	60
<i>Manual Measurements and Cursors</i> .....	60
<i>Insta-Measurements</i> .....	61
BUS DECODING OPTIONS .....	62
<i>Generic Signal Setup</i> .....	62
<i>CAN Bus Setup</i> .....	64
<i>USB Bus Setup</i> .....	65
<i>I2C Bus Setup</i> .....	66
<i>Async Bus Setup</i> .....	67
<i>Parallel Bus Setup</i> .....	68
<i>1-Wire Bus Setup</i> .....	69
<i>SPI Bus Setup</i> .....	70
<i>SDIO Bus Setup</i> .....	71
<i>SM Bus Bus Setup</i> .....	73
<i>Serial Bus Setup</i> .....	74
<i>I2S Bus Setup</i> .....	75
<i>PS/2 Bus Setup</i> .....	76
SETTING VIEWING PREFERENCES .....	77
<i>Cursor Colors</i> .....	77
<i>Background Color</i> .....	77
<i>Glass Appearance</i> .....	77
FILE OPERATIONS.....	78
<i>Creating a New File</i> .....	78
<i>Saving a Capture File</i> .....	78
<i>Open and Existing Capture File</i> .....	78
<i>Recently Used File List</i> .....	78
<i>Exporting Captured Data to a File</i> .....	78
Export Signal Data to Binary File.....	79

Export Signal Data to Text/CSV File .....	80
Export Bus Data to Text/CSV File .....	80
PRINTING .....	81
CREATING SCREEN SHOTS .....	81
SOFTWARE UPDATES .....	81
DEVELOPING YOUR OWN CUSTOM DECODERS .....	82
<i>Using the Custom Decoder</i> .....	82
<i>Building the Custom Decoder</i> .....	84
<i>Example Class Library Code</i> .....	85
Custom Decoder Parameters.....	88
Accessing Sample Data to Perform Decode.....	89
<i>Outputting Entries that will get displayed on the Screen</i> .....	89
<i>Changing the Background Color of Outputted Entries</i> .....	90
SMART SEARCH .....	91
<i>Adding a Search Line</i> .....	91
<i>Viewing Search Matches</i> .....	92
<i>Entering a Smart Search</i> .....	93
Digital Signal Edges .....	94
Analog Signal Edges .....	95
Bus Data.....	96
Digital Signal States and Ranges .....	97
Analog Signal States and Ranges .....	99
Time Window Qualifier.....	101
FAST PAN BUS VIEWING.....	104
SAMPLE AND SMART MARKERS .....	105
<i>Sample Markers</i> .....	105
<i>SMart Markers</i> .....	106
ANNOTATIONS AND STICKY NOTES .....	107
<i>Annotations</i> .....	108
<i>Sticky Notes</i> .....	109
ACQUISITION CONTROL .....	110
DISPLAY MODES .....	111
ANALOG CHANNELS SCALING .....	113
BROWSER-LIKE NAVIGATION .....	114
RELATIVE TIME DECODE .....	115
PACKETPRESENTER™ .....	117
<i>Overview</i> .....	117
<i>Setting Up the PacketPresenter</i> .....	120
<i>Viewing the PacketPresenter Output</i> .....	120
<i>Saving PacketPresenter Data to Text or RTF Files</i> .....	121

<i>Copying PacketPresenter Output to Other Programs .....</i>	<i>122</i>
<i>Searching For Packets .....</i>	<i>123</i>
<i>Filtering Packets.....</i>	<i>124</i>
<i>PacketPresenter to Waveform Association using Cursors .....</i>	<i>125</i>
<i>PacketPresenter Definition File Format .....</i>	<i>126</i>
Comments in the PacketPresenter Definition File .....	126
Constants in the PacketPresenter Definition File .....	126
PacketPresenter Definition File Sections .....	127
Protocol Section .....	127
Byte-wise busses vs. Bit-wise busses .....	127
Bus Events.....	128
Data Channels and Multiple Data Signals .....	130
Packet Section .....	131
Start and End Sections .....	131
type = Next .....	131
type = Signal .....	132
type = Value .....	132
type = Length.....	132
type = Event .....	133
type = Timeout .....	133
CHANNELX, CHANNELY or CHANNELXorY .....	134
Decode Section .....	134
Substitutions .....	135
Fields Section .....	135
Field Lines Processing .....	135
Unconditional Field Lines .....	136
Conditional Field Lines .....	136
Field Line Format .....	136
Field Format.....	136
Bus Events in the middle of a packet .....	137
Lookup Tables .....	138
Examples of Field Lines and Fields .....	139
Just Plain Data .....	139
Conditional Packet Format.....	139
String Lookup .....	140
Conditional Route of data to another Protocol .....	140
<i>PacketPresenter Add-In API .....</i>	<i>142</i>
Sample PacketPresenter Add-In Decoders .....	143
Loopback Decoder.....	143
Inverting Decoder.....	143
Expanding Decoder .....	143
Compressing Decoder .....	144

Multiple Decoders .....	145
<i>PacketPresenter Definition File Debugging</i> .....	145
<i>PacketPresenter Specifications</i> .....	146
<i>Example Protocol Files and Output Examples</i> .....	147
Async Protocol Example.....	147
I2C Protocol Example.....	148
SPI Protocol Example .....	149
CAN Protocol Example .....	151
SDIO Protocol Example .....	151
1-Wire Protocol Example.....	154
Parallel Protocol Example .....	155
Serial Protocol Example .....	156
USB Protocol Example .....	157
PS2 Protocol Example .....	159
<b>USBEE RX ANALOG SIGNAL GENERATOR .....</b>	<b>160</b>
<b>USBEE RX LOGIC PROBE AND DIGITAL VOLTAGE METER .....</b>	<b>162</b>
<b>USBEE DIGITAL LOGIC THRESHOLD.....</b>	<b>163</b>
<b>USBEE RX DIGITAL SIGNAL GENERATOR.....</b>	<b>164</b>
PULSE WIDTH MODULATORS .....	164
ARBITRARY DIGITAL PATTERN GENERATOR.....	166
<i>Specifying the Pattern With a CSV File</i> .....	166
<i>Sample Clocking</i> .....	167
<i>Waveform Generation Control</i> .....	167
<b>USBEE RX EDGE/PULSE COUNTERS.....</b>	<b>169</b>
<b>USBEE RX FREQUENCY COUNTERS .....</b>	<b>170</b>
<b>USBEE RX TOOLBUILDER .....</b>	<b>171</b>
OVERVIEW .....	171
<i>System Software Architecture</i> .....	171
THE USBEE RX POD HARDWARE .....	172
INSTALLING THE USBEE RX TOOLBUILDER.....	173
<i>USBee RX Toolbuilder ProjectS</i> .....	173
USBEE RX TOOLBUILDER FUNCTIONS .....	174
<i>Initializing the USBee RX Pod</i> .....	174
InitializeRX .....	174
<i>Reading and Setting Signal States</i> .....	174
SetSignalsRX - Setting the USBee RX Output Signals .....	174

GetSignalsRX - Reading the USBee RX Input Signals .....	175
GenerateAnalogWaveformRX – Controlling the AOUT signal.....	175
<i>Logic Analyzer and Oscilloscope Functions .....</i>	<i>176</i>
SetLogicThresholdRX – Setting the Digital Logic Threshold .....	176
StartCaptureRX .....	176
CaptureStatusRX.....	177
StopCaptureRX.....	178
TriggerNowRX .....	178
EndCaptureRX.....	178
SampleData.....	179
FindNextEdge.....	179
SetTriggersRX.....	179
DecodeUSB .....	180
DecodeSPI .....	182
DecodeI2C.....	183
DecodeCAN.....	184
Decode1Wire .....	185
DecodeParallel.....	186
DecodeSerial .....	187
DecodeASYNC .....	188
DecodeSetName .....	189
<i>Digital Pattern Generator Functions .....</i>	<i>189</i>
GeneratePWMWaveformRX.....	190
LoadDigitalWaveformRX.....	190
GenerateDigitalWaveformRX .....	190
<i>Frequency and Pulse Counter Functions .....</i>	<i>191</i>
ClearCountsRX .....	191
EnableCountsRX.....	191
DisableCountsRX.....	192
GetFrequencyAndCountsRX .....	192
EXAMPLE C CODE .....	193
<b>USBEE RX DATA EXTRACTOR OVERVIEW .....</b>	<b>198</b>
DATA EXTRACTOR FEATURES .....	198
BUS TYPES DECODED .....	199
YOUR TESTING SYSTEM .....	199
SYSTEM REQUIREMENTS.....	199
SYSTEM SETUP .....	200
<i>Running the Command Line Extractors.....</i>	<i>200</i>
<i>Building Your Own Programs Using the API.....</i>	<i>200</i>
<b>ASYNC DATA EXTRACTOR .....</b>	<b>201</b>

ASYNC BUS DATA EXTRACTOR SPECIFICATIONS .....	201
HARDWARE SETUP .....	201
EXTRACTOR COMMAND LINE PROGRAM.....	201
<i>Example Output Files</i> .....	203
EXTRACTOR API.....	207
<i>DLL filename:</i> .....	207
<i>DLL Exported Functions and parameters</i> .....	207
<i>Extraction Data Format</i> .....	210
<b>PARALLEL BUS DATA EXTRACTOR .....</b>	<b>211</b>
PARALLEL BUS DATA EXTRACTOR SPECIFICATIONS .....	211
HARDWARE SETUP .....	211
EXTRACTOR COMMAND LINE PROGRAM.....	212
<i>Example Output</i> .....	214
EXTRACTOR API.....	215
<i>DLL filename:</i> .....	215
<i>DLL Exported Functions and parameters</i> .....	215
<i>Extraction Data Format</i> .....	218
<b>SERIAL BUS DATA EXTRACTOR .....</b>	<b>219</b>
SERIAL BUS DATA EXTRACTOR SPECIFICATIONS .....	219
HARDWARE SETUP .....	219
EXTRACTOR COMMAND LINE PROGRAM.....	219
EXTRACTOR API.....	221
<i>DLL filename:</i> .....	221
<i>DLL Exported Functions and parameters</i> .....	221
<i>Extraction Data Format</i> .....	223
<b>I2C DATA EXTRACTOR .....</b>	<b>224</b>
I2C DATA EXTRACTOR SPECIFICATIONS .....	224
HARDWARE SETUP .....	224
EXTRACTOR COMMAND LINE PROGRAM.....	224
EXTRACTOR API.....	225
<i>DLL filename:</i> .....	225
<i>DLL Exported Functions and parameters</i> .....	226
<i>Extraction Data Format</i> .....	228
<b>SM BUS DATA EXTRACTOR.....</b>	<b>230</b>
SM BUS DATA EXTRACTOR SPECIFICATIONS .....	230

HARDWARE SETUP .....	230
EXTRACTOR COMMAND LINE PROGRAM.....	230
EXTRACTOR API.....	231
<i>DLL filename:</i> .....	232
<i>DLL Exported Functions and parameters</i> .....	232
<i>Extraction Data Format</i> .....	234
<b>SPI DATA EXTRACTOR .....</b>	<b>236</b>
SERIAL BUS DATA EXTRACTOR SPECIFICATIONS .....	236
HARDWARE SETUP .....	236
EXTRACTOR COMMAND LINE PROGRAM.....	236
EXTRACTOR API.....	238
<i>DLL filename:</i> .....	238
<i>DLL Exported Functions and parameters</i> .....	238
<i>Extraction Data Format</i> .....	240
<b>1-WIRE DATA EXTRACTOR.....</b>	<b>241</b>
1-WIRE BUS DATA EXTRACTOR SPECIFICATIONS .....	241
HARDWARE SETUP .....	241
EXTRACTOR COMMAND LINE PROGRAM.....	241
EXTRACTOR API.....	242
<i>DLL filename:</i> .....	242
<i>DLL Exported Functions and parameters</i> .....	243
<i>Extraction Data Format</i> .....	244
<b>I2S DATA EXTRACTOR .....</b>	<b>246</b>
I2S BUS DATA EXTRACTOR SPECIFICATIONS .....	246
HARDWARE SETUP .....	246
EXTRACTOR COMMAND LINE PROGRAM.....	246
EXTRACTOR API.....	247
<i>DLL filename:</i> .....	248
<i>DLL Exported Functions and parameters</i> .....	248
<i>Extraction Data Format</i> .....	250
<b>LOW AND FULL SPEED USB DATA EXTRACTOR .....</b>	<b>251</b>
USB DATA EXTRACTOR SPECIFICATIONS .....	251
HARDWARE SETUP .....	251
EXTRACTOR COMMAND LINE PROGRAM.....	251
EXTRACTOR API.....	252



<i>DLL filename:</i> .....	252
<i>DLL Exported Functions and parameters</i> .....	252
<i>Extraction Data Format</i> .....	255
<b>CAN DATA EXTRACTOR</b> .....	<b>260</b>
CAN DATA EXTRACTOR SPECIFICATIONS.....	260
HARDWARE SETUP .....	260
EXTRACTOR COMMAND LINE PROGRAM.....	260
EXTRACTOR API.....	261
<i>DLL filename:</i> .....	262
<i>DLL Exported Functions and parameters</i> .....	262
<i>Extraction Data Format</i> .....	265
<b>GETTING HELP</b> .....	<b>266</b>



## INTRODUCING THE USBEE RX POD



### The Tools You Need - All In One Place

Introducing the Ultimate PC-Based Programmable Multifunction Mixed Signal Oscilloscope - Logic Analyzer with I2C, SPI, Async, SDIO, 1-Wire, CAN protocol decoders, Digital Signal Generators, Frequency Counter and integrated Protocol Analyzers in a compact and easy to use USBee Test Pod.

### Designed to Solve Your Toughest Problems

Using patented technology, the USBee RX has 2 analog and 18 digital channels, 100MHz to 600MHz sampling depending on channels used, 512 million bit internal buffers, Dual 10-bit ADC's, independent 100Mps 8 channel Digital Signal Generator, 7Mps Analog Signal Generator, +/- 60V protection on all inputs, Variable Logic Thresholds and USB over current protection. And that's just the hardware.

### Mixed-Signal Multi-level Protocol Decoding

Combined with the USBee RX Suite software, the USBee RX becomes a multi-layer protocol decoder for many of today's serial and parallel busses, letting you debug at the waveform, bus data, or packet level.

### In-Field Upgradable System Supports Addition of New Features

Based on a proprietary design that allows the software, firmware and hardware to be dynamically reconfigured using web-based software downloads, the USBee RX and its ever growing list of Applications ensures that you will always have the most current tools available for years to come.

With a single USB connection to your laptop or PC, the USBee RX gives you the power to design, prototype, test, and validate your mixed signal electronic designs with seamless ease. Built upon the popular, award-winning and patented USBee mixed signal Oscilloscope, Logic Analyzer, Signal Generator, and Protocol Analyzers, the USBee RX takes full advantage of the powerful debug capabilities of the USBee RX Suite software to solve your problems fast!

The USBee RX Test Pod does not need an external power supply. The USB bus supplies the power to the pod, so your PC will be supplying the power. The Pod does, however, require a self powered hub (not bus powered) if a hub is used between the PC and Pod.



## THE USBEE RX TEST POD



The USBee RX Test Pod has four sets of connectors:

- USB
  - Connects to the PC via 6 foot USB cable
- CH1 and CH2
  - Analog Input Channels
  - BNC connectors for standard oscilloscope probes
  - +/- 60V tolerant
  - 100MHz bandwidth
- Outputs connector
  - 11 pins – 1 Ground (GND), 1 Analog Aout, 1 +5V Supply, and 8 Digital Outputs (D0-D7)
  - 0.1" spaced 0.025" square header pins
  - 8 Digital output pins (0-3.3V logic) – D0 – D7
  - +5V output – actually VBus connected directly to the USB VBus signal from the PC through a resettable fuse
  - 1 Aout analog output pin used by the Analog Signal Generator function (0-3V levels)
  - Connects to 11x1 pin test lead set
- Digital Inputs Connector
  - 20 pins (10x2) – 2 Ground (G), Clock Input (C), Trigger Input (T), and 16 Digital inputs (0-F)
  - 0.1" spaced 0.025" square header pins
  - +/-60V tolerant inputs
  - Connects to 10x2 pin test lead set

## WARNING

**IMPORTANT! - The USBee Test Pod can only be connected to a target circuit which has the same ground reference level as your PC.**

The USBee is NOT galvanically isolated. This mainly concerns systems where the target circuit AND the PC are plugged into AC power outlets. If your target system OR the PC (Laptop) are battery powered, there is no issue. If your PC and target circuit have different ground reference levels, connecting them together using the USBee GND signal can damage the devices.

To ensure both your PC and target system share the same ground reference, do the following:

1. Use polarized power cords for both the PC and target and plug them into the same AC circuit.

If you use non-polarized power cords or use separate power circuits, the PC and target system may have different ground references which can damage the USBee, target and/or PC.

2. Ensure that a GND signal on the USBee is connected to the target ground (and not another voltage level).

Also,

As with all electronic equipment where you are working with live voltages, it is possible to hurt yourself or damage equipment if not used properly. Although we have designed the USBee RX pod for normal operating conditions, you can cause serious harm to humans and equipment by using the pod in conditions for which it is not specified.

Specifically:

- ALWAYS connect at least one GND line to your circuits ground
- NEVER connect the digital signal lines (0 thru 7, TRG and CLK) to any voltage other than between -60 to +60 Volts
- NEVER connect the analog signal lines (CH1 and CH2) to any voltage other than between -60 and +60 Volts
- The USBee RX actively drives Pod signals D0 through D7 and AOUT. Make sure that these pod test leads are either unconnected or connected to signals that are not also driving. Connecting these signals to other active signals can cause damage to you, your circuit under test or the USBee RX test pod, for which CWAV is not responsible.
- Plug in the USBee RX Pod into a powered PC BEFORE connecting the leads to your design.

## PC SYSTEM REQUIREMENTS

The USBee RX Test Pod requires the following minimum PC features:

- Windows® XP, Vista or 7 32-bit or 64-bit operating system
- Pentium or higher processor
- One USB2.0 High Speed enabled port. It will not run on USB 1.1 Full Speed ports.
- 32MBytes of RAM
- 125MBytes of Hard disk space
- Internet Access (for software updates and technical support)

## EACH PACKAGE INCLUDES

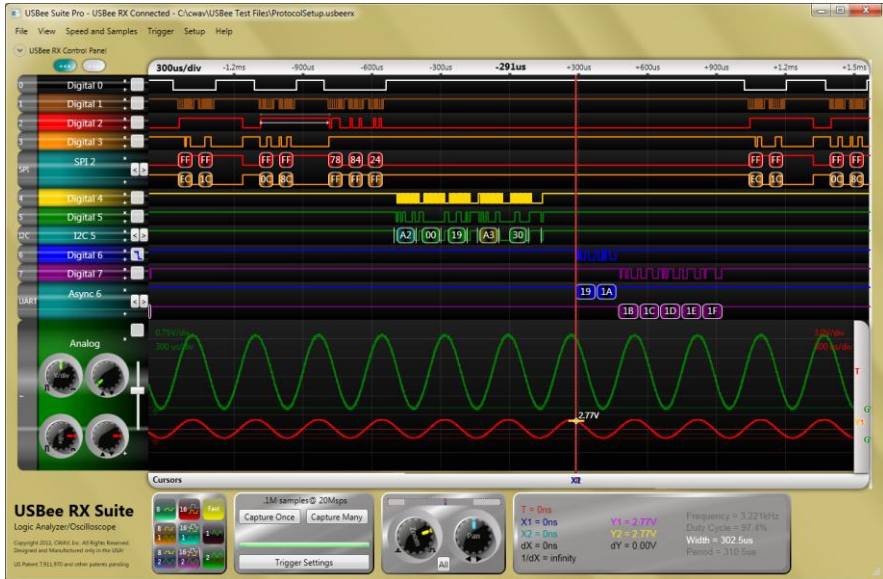
The USBee RX contains the following in each package:

- USBee RX Test Pod
- 2 100MHz oscilloscope probes with x1 and x10 selection
- 11 lead 9-inch signal generator cable set
- 20 lead 9-inch logic analyzer cable set
- 20 high performance micro grabber test clips
- 6 foot USB cable
- USBee RX Suite Software (downloaded)
- USBee RX and USBee RX Suite Manuals (downloaded)





## INTRODUCING THE USBEE RX SUITE



The USBee RX Suite is powerful electronic signal analysis software for your USBee RX Test Pod. It starts out as an easy to use Logic Analyzer, Oscilloscope, Signal Generator, Frequency Counter and PWM Controller and adds serial bus decoding and world class configurability that lets you solve your electronic problems quickly! This chapter details the operation of the USBee RX Test Pod running the USBee RX Suite software.

## USBEE RX SUITE OVERVIEW

The USBee RX Suite is a powerful mixed signal analyzer that on the USBee RX Test Pod. It is available for free from USBee.com and will run without restriction.

The USBee RX Suite Standard will run in Demo mode without a USBee, or on any USBee RX Test Pod. It will not work on any other USBee test Pod model.



USBee RX Control Panel Visible

## SEE THE INFORMATION YOU WANT FAST!

Setup of the USBee RX Suite is fast! Capturing the data you need to solve your problems is just as fast. You can see your design in action with just one click thanks to the easy to use trigger settings, color coded signals and automatic buffer and sample rate settings.

## DATA ACQUISITION OVER USB 2.0

Capture from 100 to 600 million samples per second for sample buffer depths of up to 16 million samples or 8 million transitions.

## SUPERIOR QUALITY DESIGN - PROFESSIONAL RESULTS

The USBee RX Suite takes full advantage of the power the USBee RX Test Pod. Each USBee RX comes with the best color coded highly flexible test leads, the best test clips and our signature small and sleek design that can fit right in your pocket. We are proud to say the entire USBee product line is designed and manufactured in the USA!

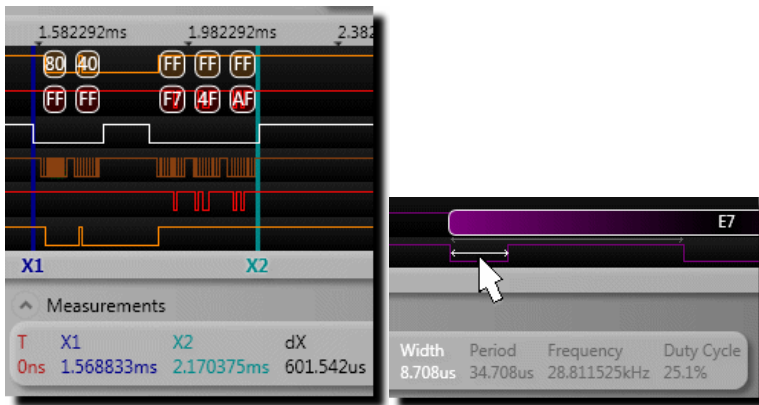
## FAST AND DETAILED WAVEFORM VIEWING

The USBee RX Suite lets you capture a huge amount of data. Go exactly to the section of that data you want using the Quick Zoom with your mouse scroll wheel, or use the Overview bar to rip through your millions of samples or hone in on a specific section.

## MEASURE IT

---

Should that pulse be 10ms? Measure it using our super easy edge snapping timing cursors. Better yet, you can use our Insta-Measure feature to instantly calculate the width, period, frequency and duty cycle of the waveform under the cursor.

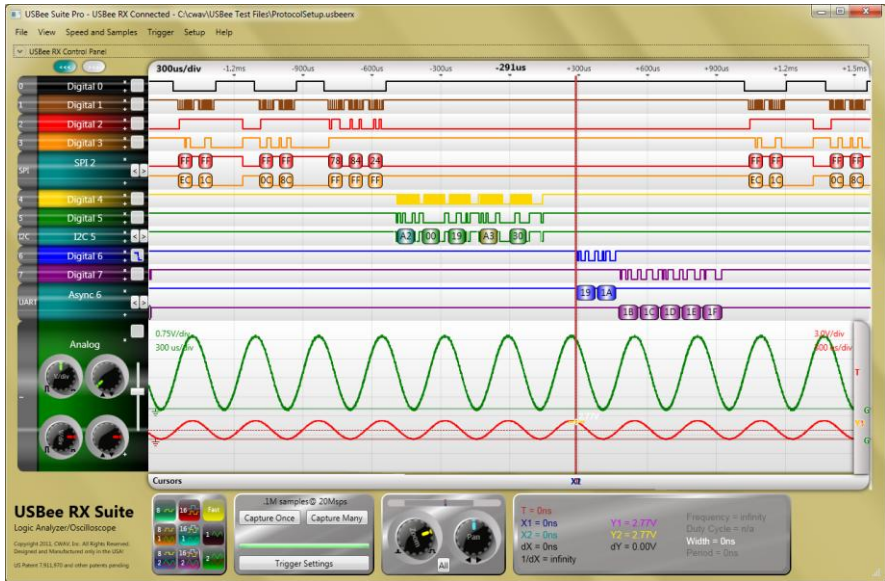


## FULLY CONFIGURABLE LOOK AND FEEL

---

View your signals like you like them. Want to add decoded bus traffic to the waveforms? Done! Want to delete waves from the screen? Done! Want to reorder waveforms for easier readability? Done! Want to resize the screen for easier reading or more data per screen? Done!

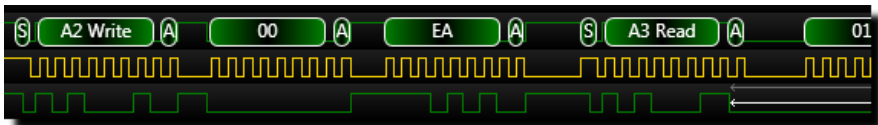
And you like Magenta? Well you can change cursor colors to suite your desires. Waveform backgrounds can also be customized, and you can even give the entire application that cool Glassy look that Vista has made so popular. Then again, if you like simple, white and black are also available. It's good that white ink cartridges are free!



USBee RX Suite showing SPI, I2C and Async decoding

## SERIAL BUS ANALYSIS

USBee RX Suite has decoding support for your favorite serial busses such as I2C, SPI and Async. Bus traffic is decoded in-line with the waveforms and can be displayed on top of, underneath, or instead of the voltage versus time waveform. Just place the cursor over the decoded traffic and get a see-through image that shows you the wiggles that made that byte!



I2C Transaction



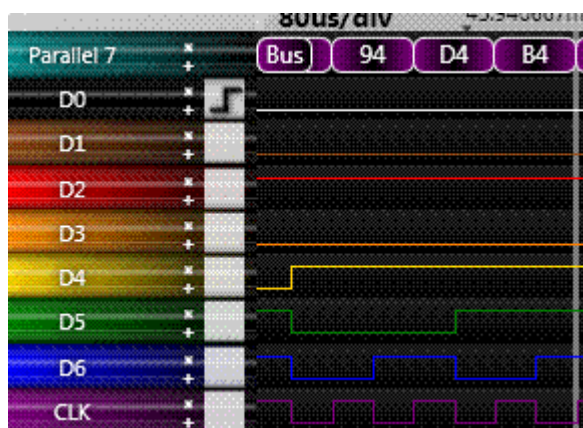
USB Transaction



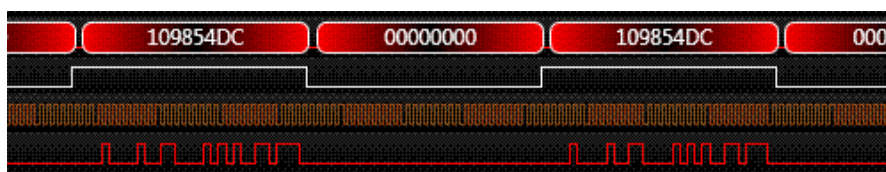
Synchronous Serial Transaction



PS2 Transaction



Parallel Bus Decode

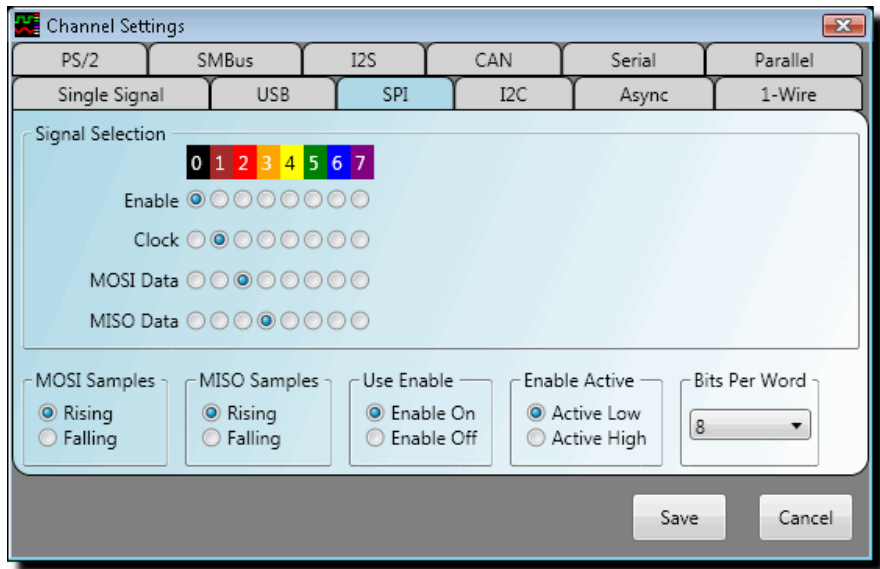


I2S Decode



CAN Transaction

Serial Bus Setup is simple and straightforward - simply choose the signals for the bus and set how the bus is configured. Not sure how your design works? Not a problem. You can try different configuration settings and the busses will be decoded using those settings on the fly so you can get it right!



(USBe SX options shown above)

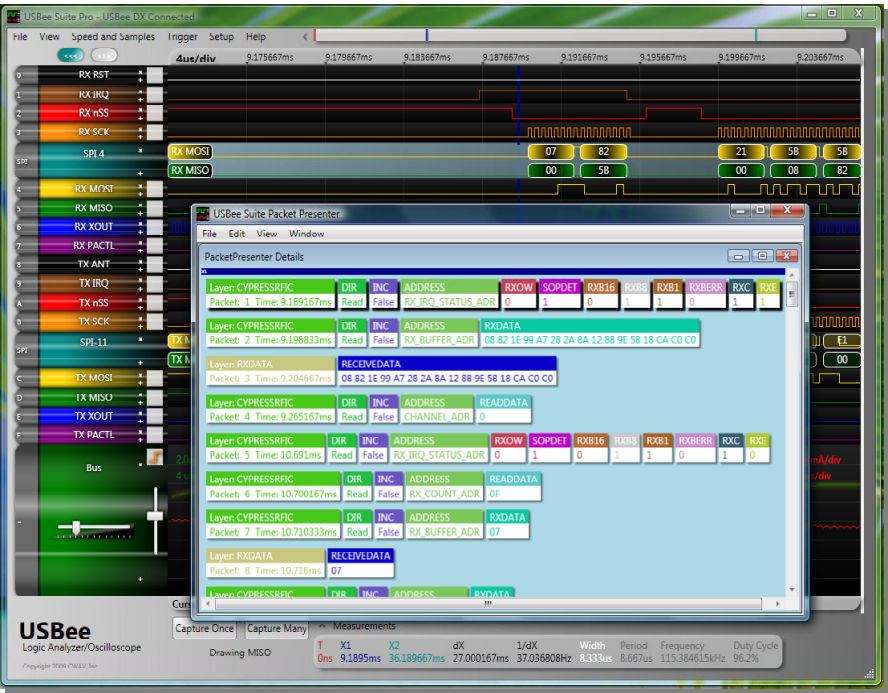
## DATA STORAGE

---

Save your entire data capture to file quickly using the USBe RX Suite data format to be read back in later for viewing. Or you can export your captured data to data files that you can work with. Want to import your waveform data into Excel? No problem! Just export it as a comma separated file and it imports directly without modification. Need the data in raw binary format? We've got that too!

PACKETPRESENTER

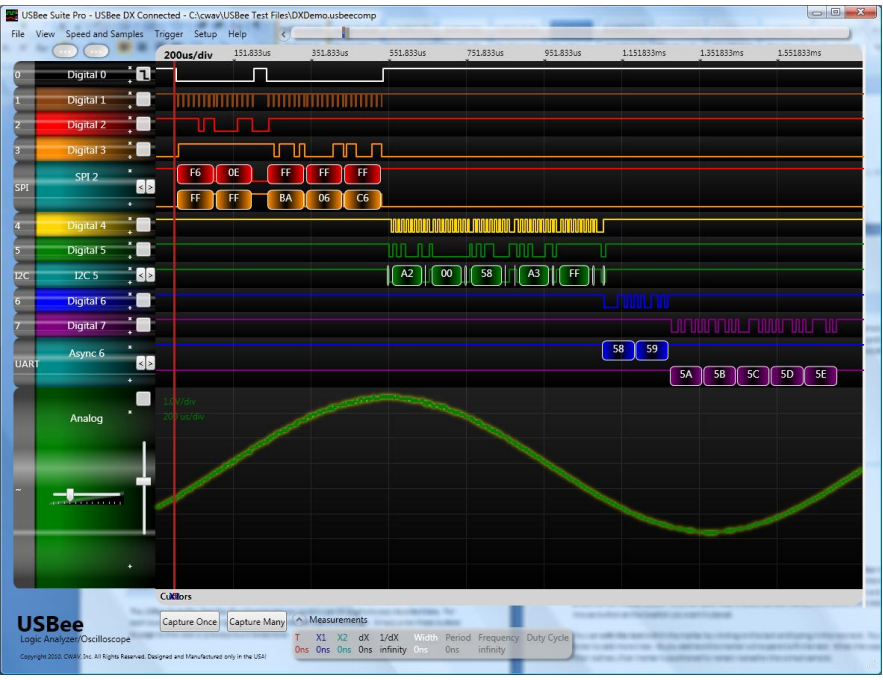
The USBee RX Suite adds the PacketPresenter™ feature that runs alongside of the existing bus decoders. The PacketPresenter™ takes the output of raw binary data from the bus decoders and parses the stream according to users PacketPresenter Definition File for the intent of displaying the communications in easily understood graphical displays.





## FAST PAN BUS VIEWING

The USBee RX Suite Fast Pan Bus Viewing lets you quickly pan through a busses decoded data. For each bus there is a left and right pan button on the left side of the screen. Simply press these buttons to page to the next or previous bus transactions.





## HOT KEYS

---

The USBee RX has a full set of hot keys that lets you drive the acquisition of samples, setup configuration and quickly view your data using the keyboard.

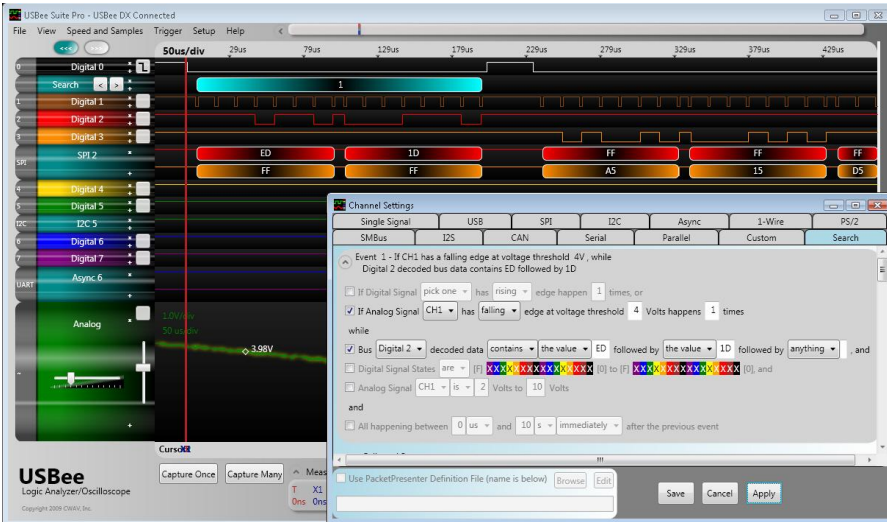
Key	Function
Left Arrow	Previous Edge/Packet
Right Arrow	Next Edge/Packet
Shift-Left Arrow	Pan Left
Shift-Right Arrow	Pan Right
Up Arrow	Previous Waveline
Down Arrow	Next Waveline
Home	Show the Start of capture buffer
End	Show the end of the capture buffer
Ins	Insert a new waveline below the current line
Del	Delete the current waveline
Space	Capture Once
1	Go to the X1 cursor
2	Go to the X2 cursor
A	Zoom out to view all of the capture buffer
B	Open the Waveline settings dialog box
D	Toggle the Decoder List window
I	Zoom In
O	Zoom Out
T	Go to the Trigger position

# SMART SEARCH

USBee RX Suite Smart Search highlights the sections of your trace matching your areas of interest so that you don't need to waste time hunting for the data you need.

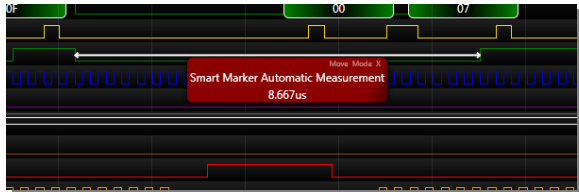
You can specify up to 32 levels of search events that are any combination of bus decoded traffic, states or edges of digital or analog signals, inside or outside of analog voltage ranges and/or digital ranges, and all validated by time specific windows.

Once specified you can pan through the occurrences of your searched items with the click of the mouse and see the total number of times the searched events occur.



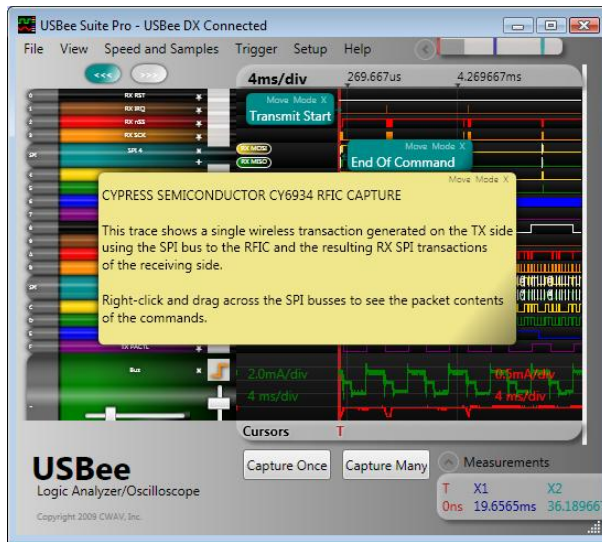
# SAMPLE AND SMART MARKERS

Placing markers in your traces can help detail what is happening in your design. There are two types of markers that can be used. The first marker type locks itself to a sample on a waveform and lets you specify the text. The second is a Smart Marker that automatically measures the pulse width, frequency, period or duty cycle of the waveform at the marker location.



## ANNOTATIONS AND STICKY NOTES

The USBee RX Suite adds Sticky Notes which you can use to further detail your traces for documentation purposes. You can also add Title and Footer text to your display that is saved with the trace file.



## ACQUISITION CONTROL

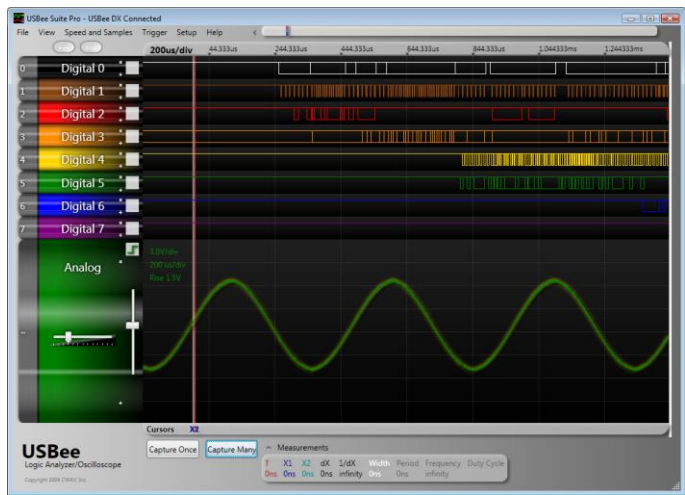
The USBee RX Suite adds more trace acquisition and triggering controls such as Normal Mode, Automatic Mode, Single Capture and Multiple Capture.

Normal mode will wait for the trigger event to occur before capturing. Automatic Mode will wait a set time for the trigger and will automatically trigger if it is not found.

Single Capture mode performs a one-shot capture of the signals. Multiple Capture repeatedly captures and displays the signals.

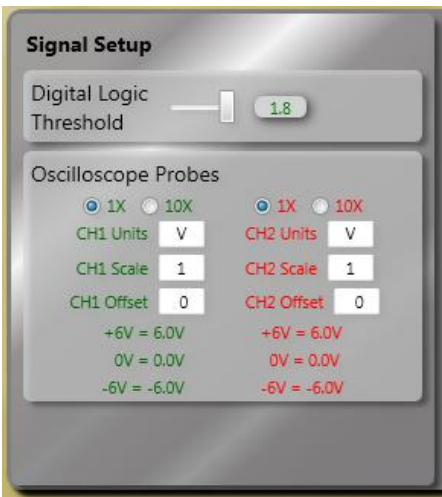
## DISPLAY MODES

The USBee RX Suite lets you widen the trace waveforms, display the analog waveforms as vectors or single sample points, and persist the display from one trace to the next.



## ANALOG CHANNELS SCALING

The USBee RX Suite provides a scaling ability to convert the analog voltages into other units of measurement.

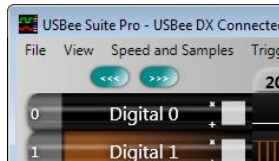


## USBEE SUITE DATA FILE IMPORTING

The USBee RX, AX or ZX running the USBee Suite software saves files in their own file format. These older files can be imported into the USBee RX Suite.

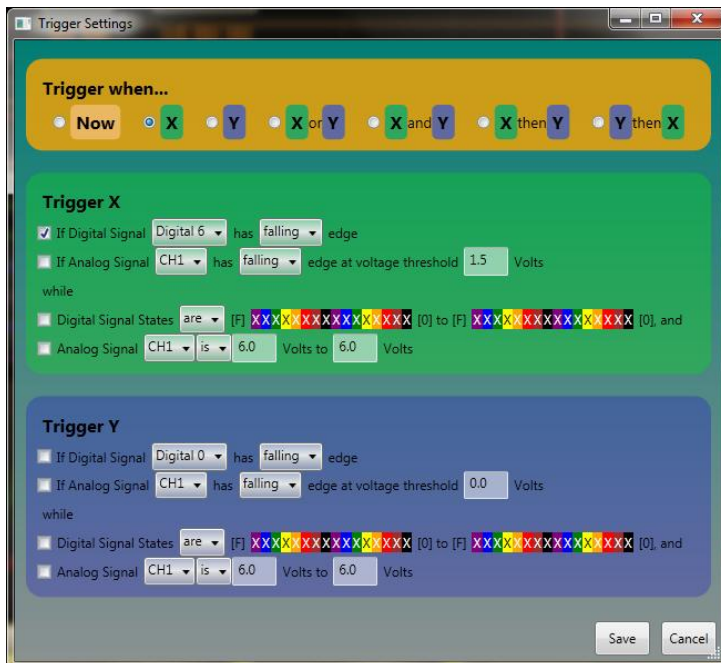
## BROWSER-LIKE NAVIGATION

The USBee RX Suite adds browser-like Forward and Back buttons that let you quickly navigate through your trace display.



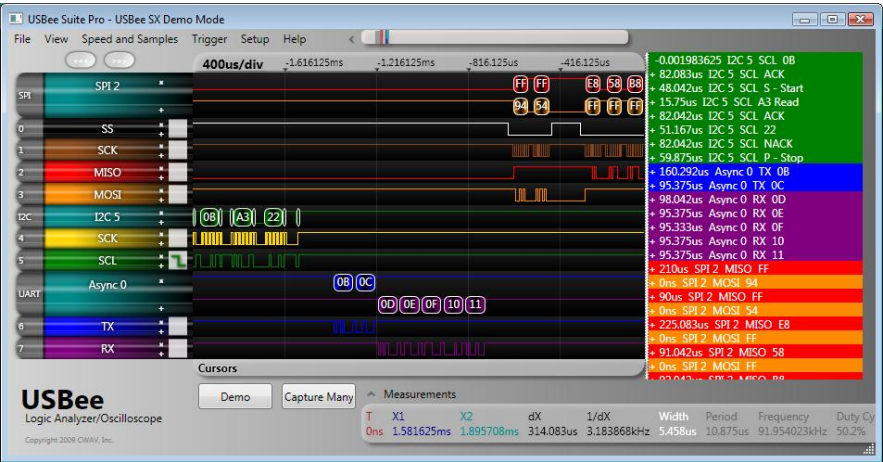
## COMPLEX TRIGGERING

The USBee RX Suite adds a multi-level hardware trigger to capture the events you need to see to solve your problems.



RELATIVE TIME DECODE

The USBee RX Suite also adds a Relative Time or Absolute Time setting for the decoded data lists.



## USBEE RX CONTROL PANEL

The USBee RX Suite contains a USBee RX Control Panel that controls many of the features of the USBee RX system, including the Variable Logic Threshold, Analog channel scaling, Current Logic and Analog levels, Analog Signal Generator, Digital Signal Generator, PWM controllers, Pulse Counters and Frequency Counters.



## QUICK START

This section goes through installing the software for your USBee RX and getting you using the USBee RX quickly.

## PC SYSTEM REQUIREMENTS

The USBee RX Suite requires the following minimum PC features:

- Windows® XP SP3, Vista or Windows 7 32-bit or 64-bit operating system
- .NET Framework 4.0 or greater. This is installed automatically during installation if not already on your PC.
- Pentium or higher processor
- One USB2.0 High Speed enabled port. It will not run on USB 1.1 Full Speed ports.
- 32MBytes of RAM
- 125MBytes of Hard disk space
- Internet Access (for software updates and technical support)



## SOFTWARE INSTALLATION

To ensure that you are using the latest version of USBee RX software, you can download the software from our web site at [www.usbee.com](http://www.usbee.com). You must install the software on the PC before you plug in the USBee RX device.

A quick guide to install the software follows:

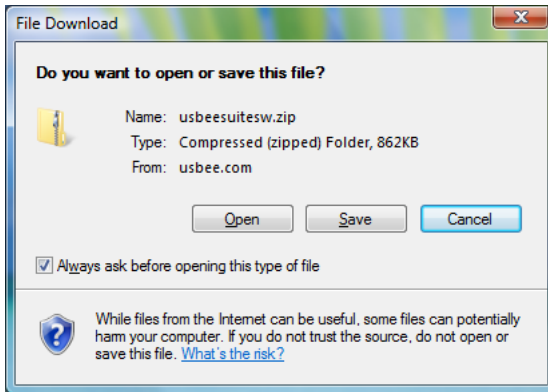
- Download the USBee RX Software from <http://www.usbee.com/download.htm> and open the usbeerxsuitesw.zip file.
- Run the SETUP.EXE.
- Follow the instructions on the screen to install the USBee RX software on your hard drive. This may take several minutes.
- Now, plug a USB A to USB Mini-B cable into the USBee RX and the other end into a free USB 2.0 High Speed port on your computer.
- You will see a dialog box indicating that it found new hardware and is installing the software for it. Follow the on screen directions to finish the driver install.
- The USBee RX Software is now installed.
- Run the USBee RX Suite software by going to the Start | Program Files | USBee RX Suite.
- If your PC does not already have .NET Frameworks version 4.0 or greater, you will be notified to download and install this from the Microsoft web site.

## DETAILED SOFTWARE INSTALLATION

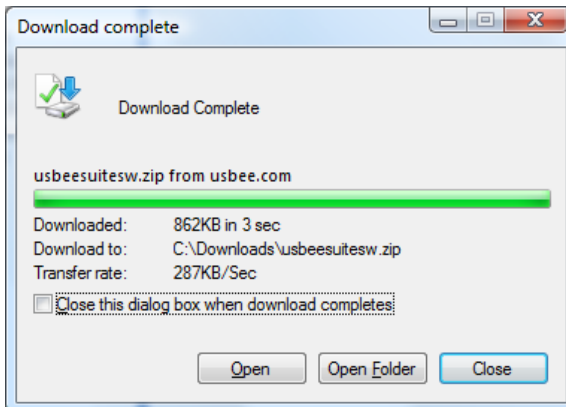
The USBee RX Suite software is available for download from [www.usbee.com/download.htm](http://www.usbee.com/download.htm). It will run in a demonstration mode if you do not have a USBee RX Pod installed and attached. To install the software for demo purposes, just install the USBee RX Suite software.

To install the USBee RX Suite software:

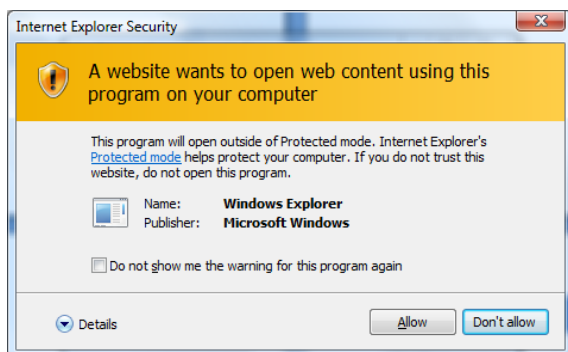
- Click the USBee RX Suite Software link at <http://www.usbee.com/download.htm> and click SAVE to save the software to a known directory.



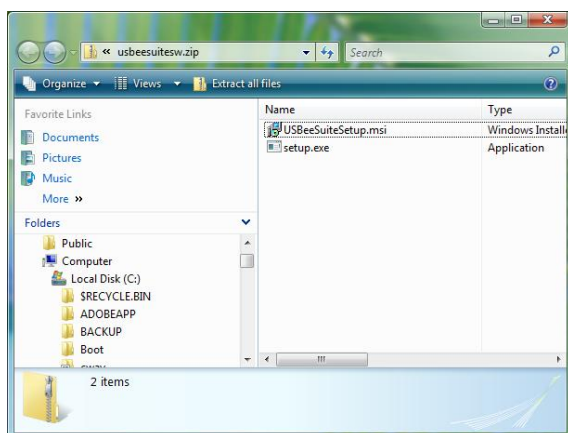
- Open the ZIP file you just downloaded by clicking OPEN.



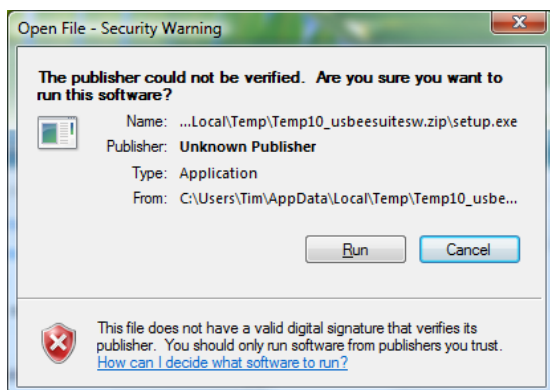
- If you receive messages such as below, press ALLOW or CONTINUE ANYWAY to continue with installation of the software.



- Run the SETUP.EXE file that is included in the ZIP file that you downloaded to start the installation..



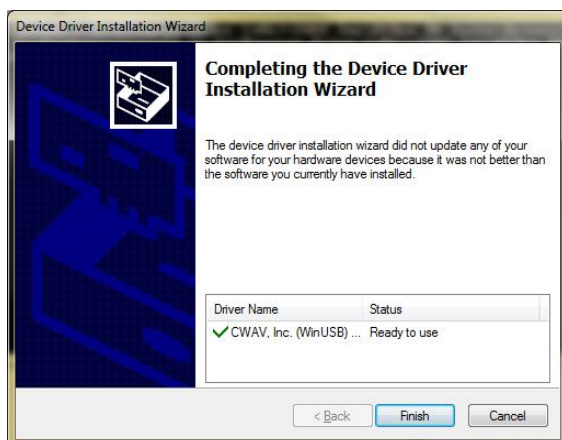
- If you get the following warning, click RUN to continue with the installation.



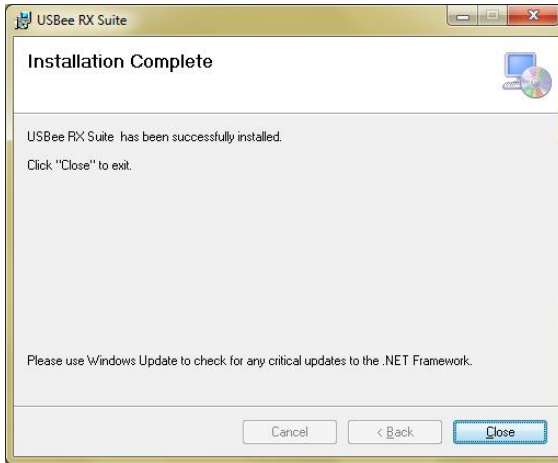
- The first part of the installation installs Microsoft requirements, including Microsoft .NET Frameworks Version 4.0. If you do not have this on your PC it will install it for you as shown below – click Accept to install the .NET Frameworks. This installation takes a LONG time, so please be patient since it is worth the wait! If you already have it installed, you will automatically see the “Welcome to the USBee RX Suite Setup Wizard” screen.
- You will see the Welcome to the USBee RX Suite Setup Wizard screen as shown below



- During the installation you will see a driver installation dialog box that will install the USBee RX drivers. Click Finish to complete the installation after the drivers have been successfully installed as below.



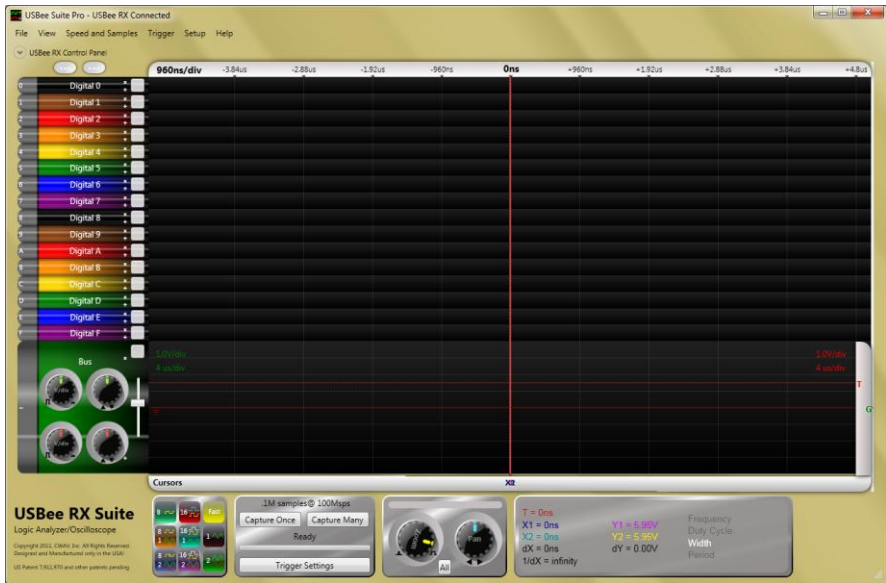
- Follow the instructions (clicking NEXT each time) on the screen to install the USBee RX Suite software on your hard drive. This may take several minutes. When completed you will see the following screen.



- Click CLOSE and the USBee RX Suite software is now installed.
- To run the USBee RX Suite software, choose the USBee RX Suite icon from the Windows Start Menu or click on the USBee RX icon on the desktop. If no USBee RX is plugged in or installed, you will see the following screen running in Demo Mode (see the top title bar).



- If you have a USBee RX plugged in and installed correctly, you will see a screen with all available channels shown. Below is the USBee RX version showing 16 digital channels and 2 analog channels. You can also see that the device is connect (and not in demo mode) in the top title bar.



## DEMO MODE

In the Demo mode, you can see an example trace capture by clicking the Demo button. This loads a trace that includes a number of serial busses and lets you see how the USBee RX Suite can decode the bus traffic, manipulate the waveform data, and use the features of the USBee RX Suite.

The USBee RX Suite display shows the USBee connection status in the title bar of the application. When a USBee RX is connected to the computer when the application starts, the title bar indicates the connection status.

If you run the software with no pod attached, it will run in demonstration mode and simulate data so that you can still see how the software functions.

If you are running in Demo mode and you want to connect to your USBee RX pod, you must exit the USBee RX Suite, connect the USBee RX and then rerun the USBee RX Suite software.

## TESTING YOUR CIRCUIT USING THE USBEE RX

In order to quickly get up and running using The USBee RX Suite application, here is a step by step list of the things you need to do to view a waveform trace, after you have installed the software and hardware.

- **Plug in the USBee RX Pod** - Plug the USBee into your computer USB High Speed port
- **Connect Ground** - Connect the GND wire to the Ground of your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to the Ground.
- **Connect Signals** - Connect any of the USBee inputs on the USBee pod to your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to your signal of choice.
- **Run USBee RX Suite** - Run the USBee RX Suite Application from the Start Menu.
- **Press the Capture Once button** - This will capture and display the current activity on all of the signals.
- **View the Waveforms** - You can then scroll the display, either by using the slider bars, or by clicking and dragging on the waveform itself. You can also change the knobs to zoom the waveform.
- **Make Measurements** - You can make simple measurements by using the Cursors area (gray bars under the waves). Click the left mouse button to place one cursor and click the right mouse button to place the second. The resulting measurements are then displayed in the Measurements section of the display.

## USING THE USBEE RX SUITE MIXED SIGNAL OSCILLOSCOPE

This section details the operation of the Mixed Signal Oscilloscope, Logic Analyzer and Protocol Analyzer features of the USBee RX Suite application that runs on the USBee RX Test Pod.

When the USBee RX Suite is first run, you will see a screen containing all of the available input signals for the USBee RX Pod plugged into the PC.



The USBee RX Suite maintains its last configuration and will reload that configuration when it is run again. This configuration is located in your `\Users\NAME\AppData\Local\USBeeSuite` directory where NAME is your username. To reset the software to the initial state you can delete the files in that directory.

## ANALYZER SETUP

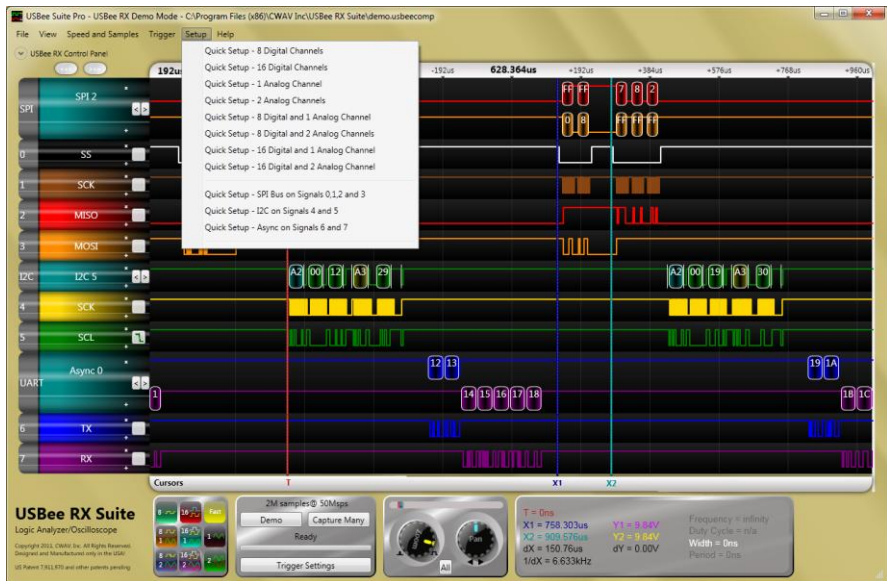
### QUICK SETUP CONFIGURATION

The USBee RX Suite can capture all of the input channels on the USBee RX. With a RX plugged in, it can capture 18 channels of digital (16 data lines, Clock and Trigger signals) and 2 channels of analog at the same time.

Although you can individually show each signal, there are a number of Quick Configurations that let you instantly select just the channels you need.

To select a configuration, click **Setup** on the menu and select the Quick Setup configuration of your choice. Below shows the available Quick Setup options for the USBee RX.





You can also select these modes using the buttons at the bottom of the screen:

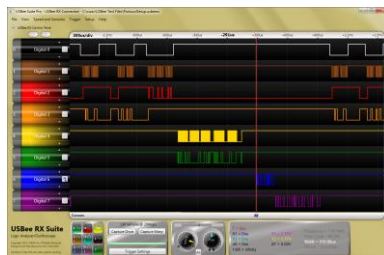


The **FAST** button sets the Sample Rate and buffer size to give the fastest screen update rates.

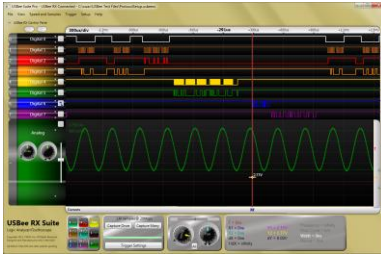
Below are examples of the application in various modes.



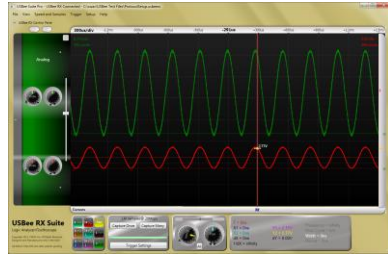
16 Digital–2 Analog Channels



8 Digital–0 Analog Channels



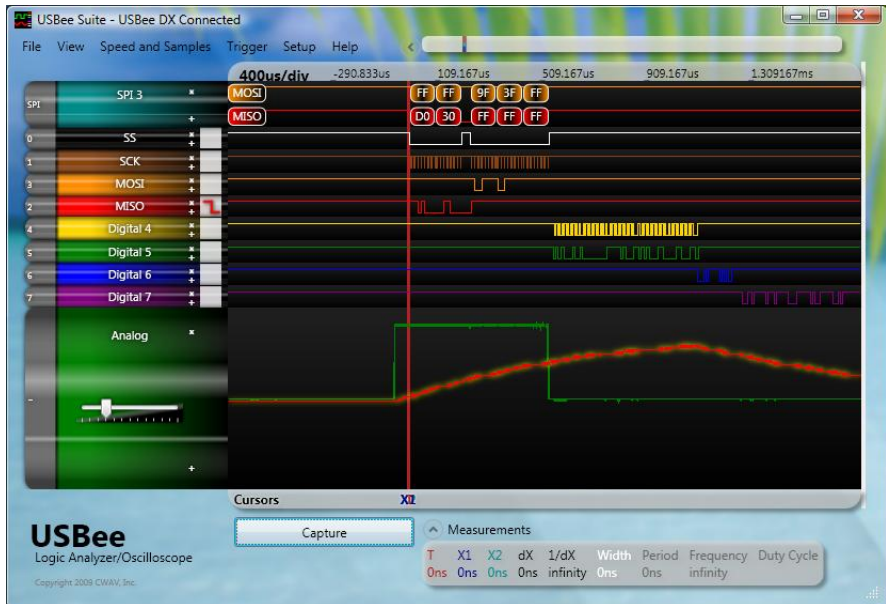
8 Digital–1 Analog Channels



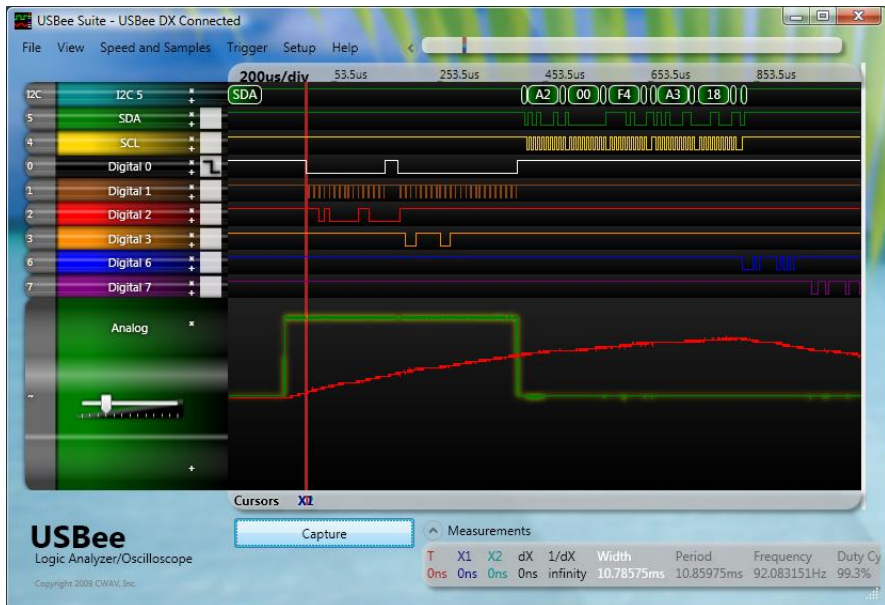
0 Digital–2 Analog Channels

There are also three other Quick Setup features that let you instantly setup an I2C, SPI or ASYNC decoder line.

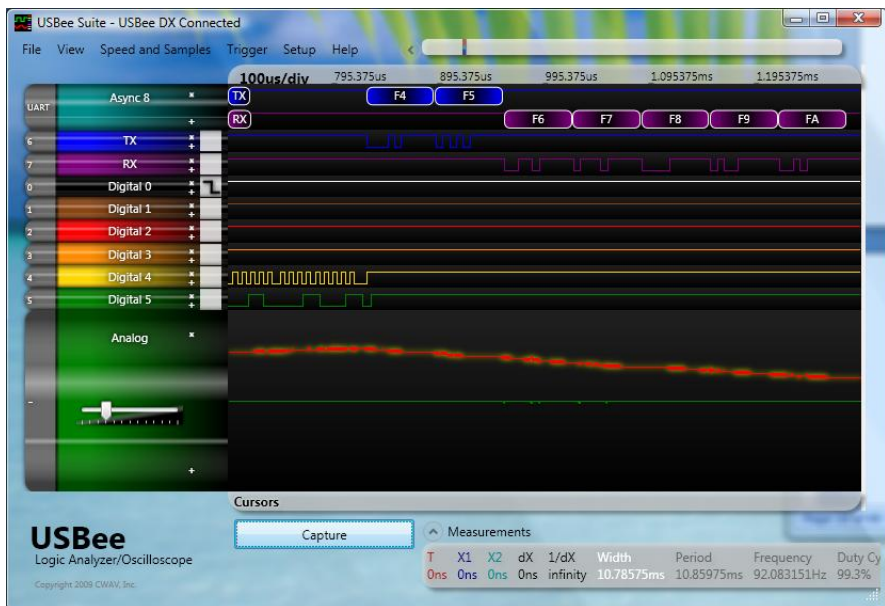
The **Quick Setup – SPI** configures the first 4 lines to be an SPI bus with the SS, SCK, MOSI and MISO lines. It also adds a decoder line to the screen with this data decoded as below.



The **Quick Setup – I2C** configures the signals 4 and 5 to be an I2C bus with the SDA and SCL lines. It also adds a decoder line to the screen with this data decoded as below.



The **Quick Setup – ASYNC** configures the signals 6 and 7 to be a full duplex ASYNC bus with the TX and RX lines. You will need to change the baud rate, data bits and parity to match your bus. It also adds a decoder line to the screen with this data decoded as below.



---

## HOT KEYS

The USBee RX has a full set of hot keys that lets you drive the acquisition of samples, setup configuration and quickly view your data using the keyboard.

Key	Function
Left Arrow	Previous Edge/Packet
Right Arrow	Next Edge/Packet
Shift-Left Arrow	Pan Left
Shift-Right Arrow	Pan Right
Up Arrow	Previous Waveline
Down Arrow	Next Waveline
Home	Show the Start of capture buffer
End	Show the end of the capture buffer
Ins	Insert a new waveline below the current line
Del	Delete the current waveline
Space	Capture Once
1	Go to the X1 cursor
2	Go to the X2 cursor
A	Zoom out to view all of the capture buffer
B	Open the Waveline settings dialog box
D	Toggle the Decoder List window
I	Zoom In
O	Zoom Out
T	Go to the Trigger position

---

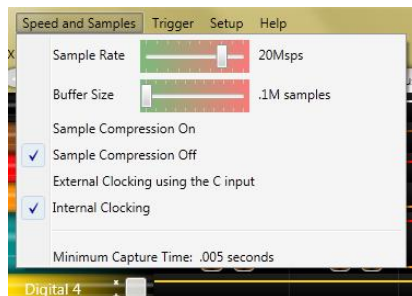
## SIGNAL NAMES

To change the names shown for a signal, click on the signal name and enter a new name.

---

## BUFFER SIZES AND SAMPLE RATE SETTINGS

The USBEE RX SUITE captures the behavior of the digital and analog signals and displays them as “traces” in the waveform window. The Speed and Samples menu lets you choose how the traces are captured. Below shows the Speed and Samples menu.



The **Buffer Size** lets you select the size of the Sample Buffer that is used. For each trace, the buffer is completely filled, and then the waveform is displayed. You can choose buffers that will capture the information that you want to see, but remember that the larger the buffer, the longer it will take to fill, display and decode.

You can also choose the **Sample Rate** that you want samples taken. You can choose from 100ksps (samples per second) to up to 600 Msp/s. At up to 100Msp/s all channels can be sampled. At higher sample rates, a subset of channels is available. The table below shows how many channels are available for the given sample rates.

	Available Channels and resulting Sample Rate for each channel																	
Sample Rate Setting in sps	CH2	CH1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
< = 100M (1)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
200M (2)	1	1	2	2	2	2	2	2	2	2	x	x	x	x	x	x	x	x
300M (3)	1	1	3	3	3	3	3	1	x	x	x	x	x	x	x	x	x	x
600M (6)	1	1	6	6	1	1	1	1	x	x	x	x	x	x	x	x	x	x

The USBee RX can use sample compression to lengthen the capture time. In **Sample Compression** mode, only the transitions are stored, effectively removing the samples between inactivity. To turn on Sample Compression, select Sample Compression On in the Speed and Samples menu.

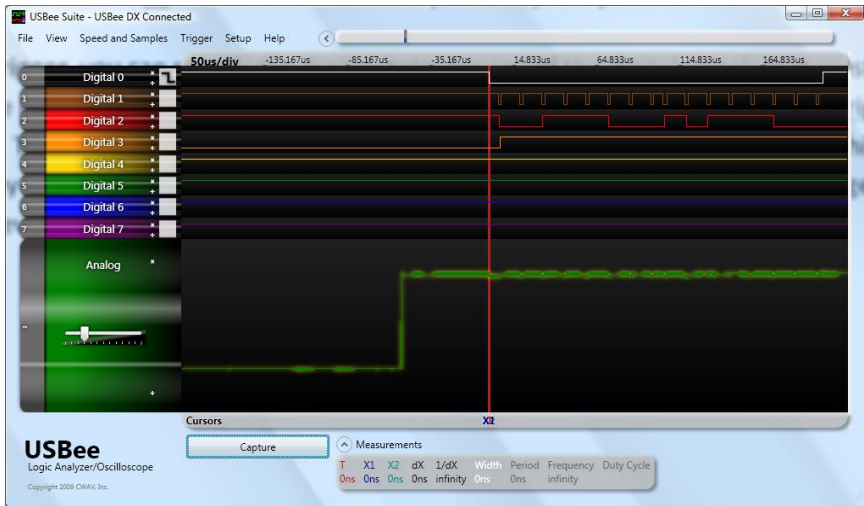
The Sample Compression works on just the channels that are shown on the screen, so for the longest trace buffers, only display the signals that you are interested in. All other channels will be masked out and not captured.

You can also use an external clock (using the C input) to gather each sample instead of the internal sample clock. To turn on **External Clocking**, select External Clocking using the C Input on the Speed and Samples menu. The external clock can run up to 100MHz. When external Clocking is enabled, Sample Compression is not available.

## SETTING SIMPLE TRIGGERS

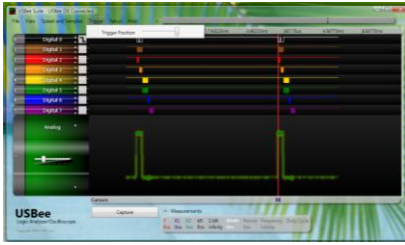
The USBee RX Suite uses a Trigger mechanism to allow you to capture just the data that you want to see.

For a **Digital trigger**, you can specify the digital states for any of the digital signals that must be present on the digital lines before it will trigger. Below shows the trigger settings (to the right of the Signal labels). This example shows that we want to trigger on a falling edge of Signal 0, which is represented by a high level followed by a low level. To change the level of any of the trigger settings, just click the level button to change from don't care to rising edge to falling edge.

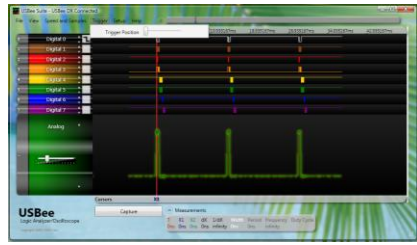


The waveforms are shown with a trigger position which represents where the trigger occurred. This sample point is marked on the waveform display with a Vertical red cursor line and a "T" in the horizontal cursors bar.

You can use the **Trigger Position** menu setting to specify how much of the data that is in the sample buffer comes before the actual trigger position. If you place the Trigger Position all the way to the left, most of the samples taken will be after the trigger sample. If you place Trigger Position all the way to the right, most of the samples taken will be before the Trigger sample. This control lets you see what actually happened way before or way after the trigger occurred.



Trigger Position to the Right



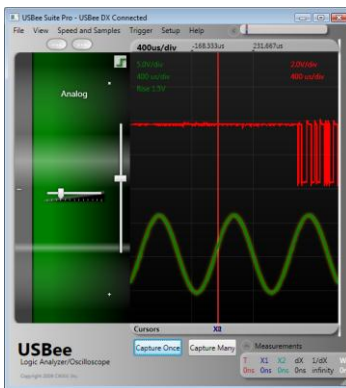
Trigger Position to the Left

For an **Analog trigger** you must specify the Channel to use, Rising or Falling Edge, and the Trigger Level. Click on the Trigger Settings Box (to the right of the waveline delete "X") repeatedly to toggle through Channel 1 Rising, Channel 1 Falling, Channel 2 Rising, Channel 2 Falling and None. You then specify the trigger voltage level (-6V to +6V) by using the vertical slider on the left hand side of the analog waveform display. The trigger level edge and value will be shown as you scroll this level underneath the Volts/Div and Secs/Div labels within the waveform area.

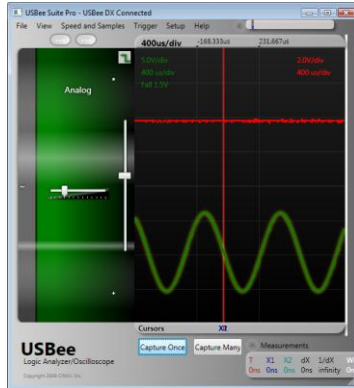


For an analog trigger, the trigger position is where the waveform crossed the **Trigger Voltage** level that you have set at the specified slope. To move the trigger voltage level, just move the slider on the left of the waveform.

The following figures show a trace captured on each of the edges.



Analog Trigger Slope = Rising Edge



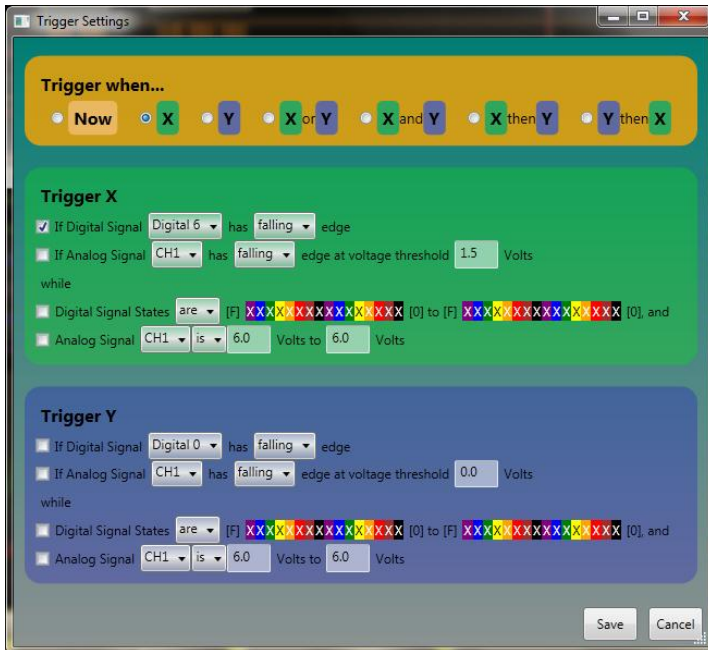
Analog Trigger Slope = Falling Edge



The Trigger position is placed where the actual signal crosses the trigger voltage with the proper slope. The USBee pods allow for huge sample buffers, which means that you can capture much more data than can be shown on a single screen. Therefore you can scroll the waveform back and forth on the display to see what happened before or after the trigger.

## SETTING COMPLEX TRIGGERS

The USBee RX Suite also allows you to set more complex triggers using the Triggers Dialog box as shown below.

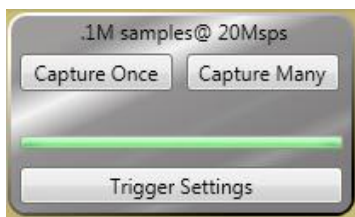


There are two levels of triggers available, X and Y. Each trigger event (X or Y) can have any combination of analog or digital edges which can be qualified using digital patterns or analog voltage levels.



## CAPTURING WAVEFORM DATA

Press the Capture buttons to start capturing the waveform data from your hardware design. If you are running in Demo mode, the button reads Demo. Capture Once performs a single capture, while Capture Many repeatedly captures and displays consecutive traces.

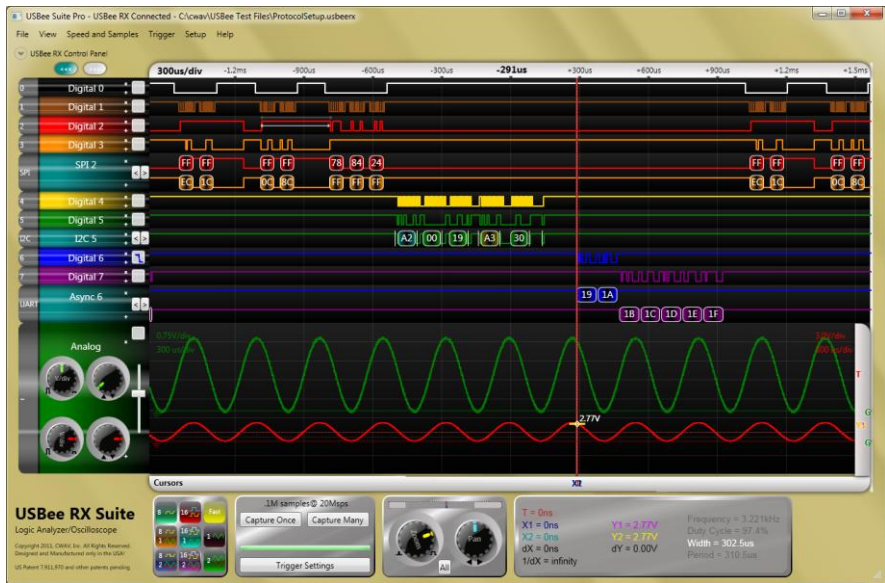


Each capture will look for the trigger condition, fill the buffer with samples of the signals and stop. If you would like to stop the capture before it is completed just press the same button again (which reads STOP during a capture). After a trace is captured, the waveform data is gathered, decoded (if needed) and displayed in the waveform window.

## VIEWING CAPTURED DATA

### SCROLLING, ZOOMING AND PANNING WAVEFORMS

The Waveform display area is where the measured signal information is shown. It is displayed with time increasing from left to right and voltage increasing from bottom to top.



The position of the waveform defaults to show the actual trigger position in the center of the screen after a capture. However, you can move the display to see what happened before or after the trigger position.



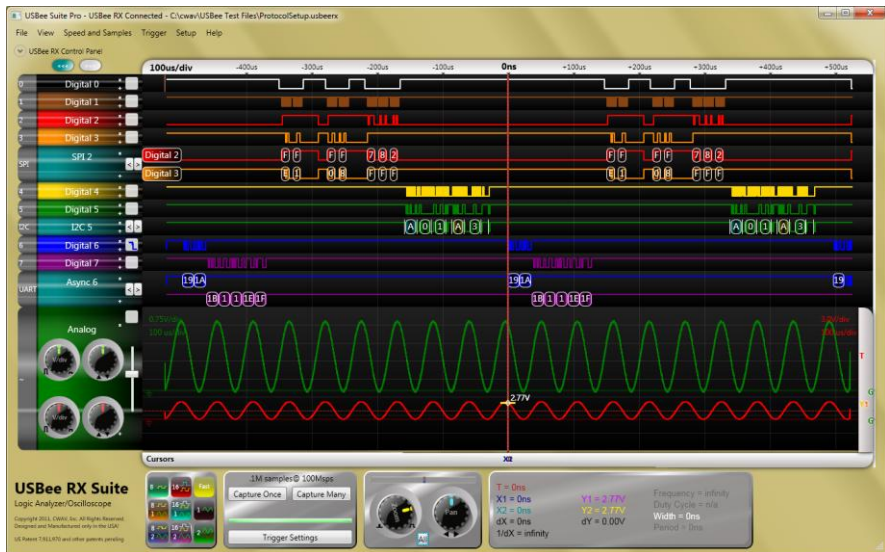
To **Scroll the Waveforms in Time** left and right, you can click on the overview bar above the knobs, scroll the **Pan knob** (click and drag or mouse wheel on the knob) or you can simply click and drag the waveform itself with the left mouse button.

To **Scroll the Analog Waveform in Voltage** up and down, you can simply click and drag the waveform itself by selecting and dragging using the mouse or use the **analog offset knob** to the left of the analog waveforms.

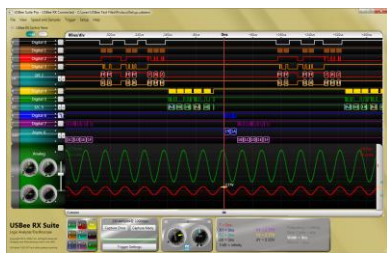
To **Zoom In** or **Zoom Out**, or other words change the number of **Seconds per Division**, you can use the scroll wheel or single click on the waveform. You can also use the **Zoom knob** (click and drag or mouse wheel on the knob). To zoom in, scroll up or click the left mouse on the waveform window. To zoom out in time, scroll down or click the right mouse button on the waveform window.

To change the number of **Volts per Division** for an analog channel, use the **V/div knob** (click and drag or mouse wheel on the knob)highlight the channel you want to change, or you can hold the left mouse button down and use the scroll wheel. You can also highlight the signal and use the slider bar to the left of the waveform.

To view the entire capture buffer on the display, press the **All** button. The screen below shows all of the collected samples on a single screen for that trace.



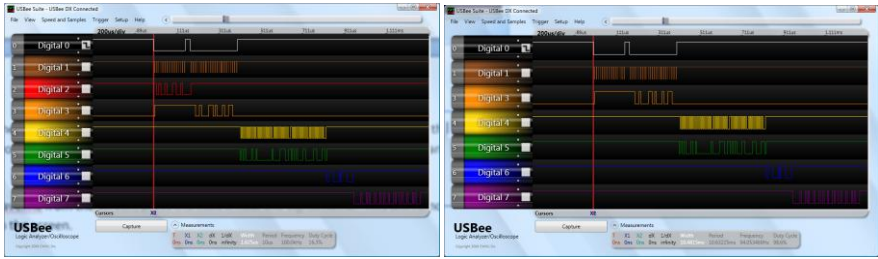
To **Stretch and Shrink** the display, you can click and drag the edges of the application to the size you want. All waves will scale to fit. Below you see two examples of different size displays.



## MODIFYING WAVE LINES

Each line on the display is called a **Waveline**. Wavelines can be modified to your liking so that you see the data you need to solve your problem. You can delete, add, move, or reconfigure any waveline.

To **Delete a Waveline** from the screen, press the little X near the signal name. This will remove the waveline from the screen. Below shows the USBee RX Suite after deleting the Digital 2 waveline.

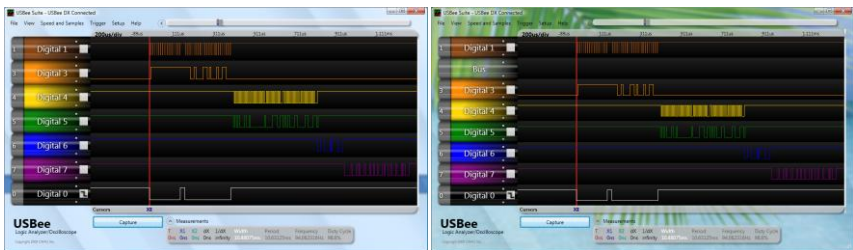


Removing a signal from the screen may not remove the signal from subsequent captures. Signals will only be eliminated from captures if all signals from a given byte lane are removed.

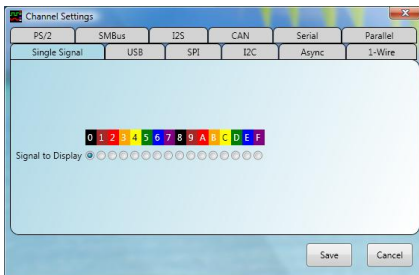
To **Move a Waveline**, simply click on the gray tab on the left and drag it to the new position. Below shows the Digital 0 signal moved to the bottom.



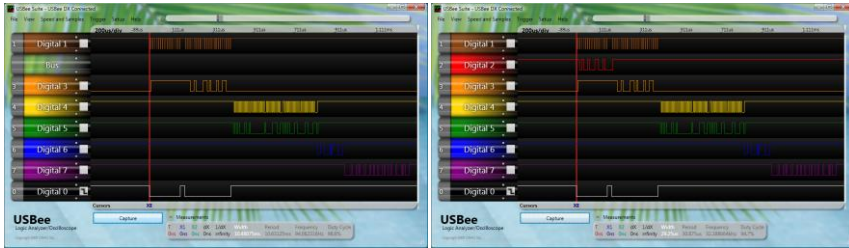
To Add a Waveline, click on the small + sign above where you want to insert the new waveline. Below you see a new waveline inserted after the first waveline.



When you insert a new waveline the Channel Selection dialog box appears for you to choose the settings for that waveline. Below shows the Channel Settings Dialog Box.



Once you select the properties of the new waveline, it will be displayed with the other signals. Below shows a new line added that shows the Digital 2 single signal.



We will go through creating Bus wavelines that decode bus traffic in-line in the next section.

To **Modify an Existing Waveline** click on the grey tab on the left of the waveline. This will bring up the Channel Settings dialog box and allow you to change the settings for that line. Below we modified the last line to show Digital 2 signal instead of the Digital 0 signal.



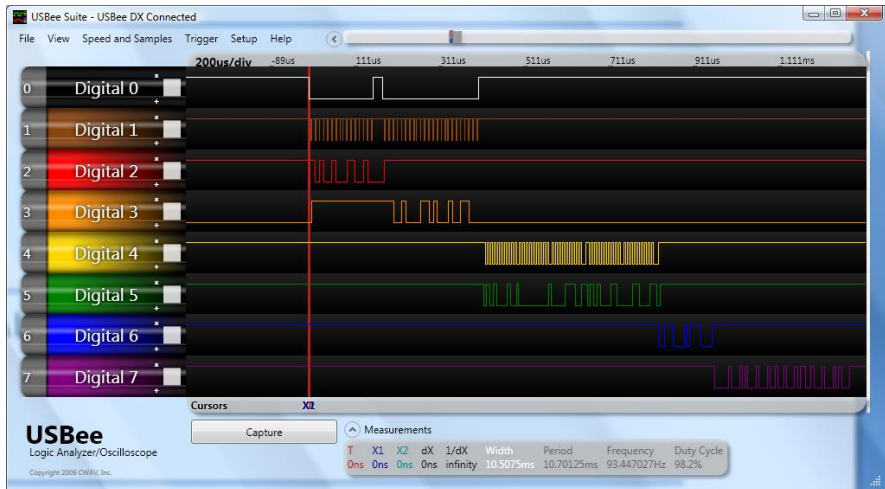
---

## DECODING BUS TRAFFIC INLINE

The USBee RX Suite software can decode certain types of serial busses automatically and display that information in-line with the waveforms. Below is an example of a screen that shows 3 different serial bus decoders at the same time, one SPI, one I2C, and one full duplex ASYNC channel.

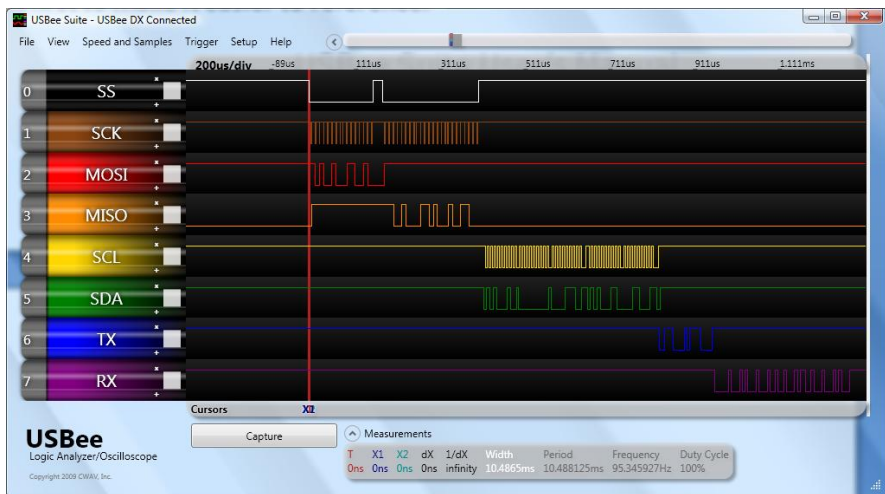


We will go through an example that shows how to setup various busses. We start with a capture of 8 digital lines that have a number of busses included.

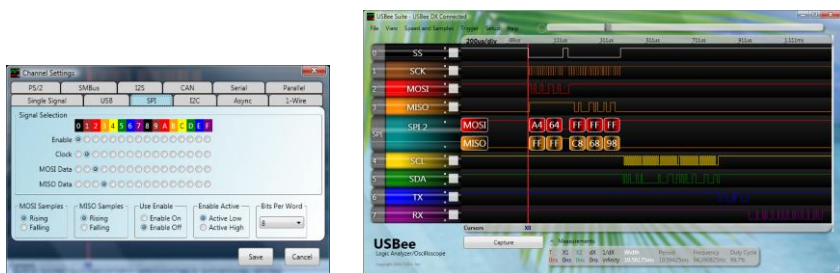


In this example, we will name the bus signals first to make it easier to reference.



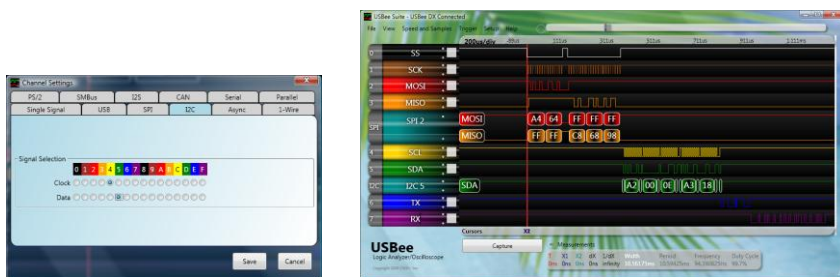


Now we will add an SPI bus which is made up of the first 4 signals. We press the small + sign near the MISO label to insert the waveline below that line. We then get the Channel Settings Dialog and choose the SPI tab. The following screen is then displayed.

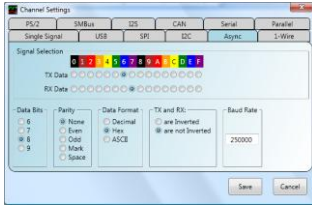


From here we select the parameters for this bus (shown above) and press Save. Once we press Save, the line is added to the screen, the current trace is decoded, and the decoded information is shown on the waveline.

We then add the I2C bus using the Channel Settings dialog box as below with the resulting waveline.



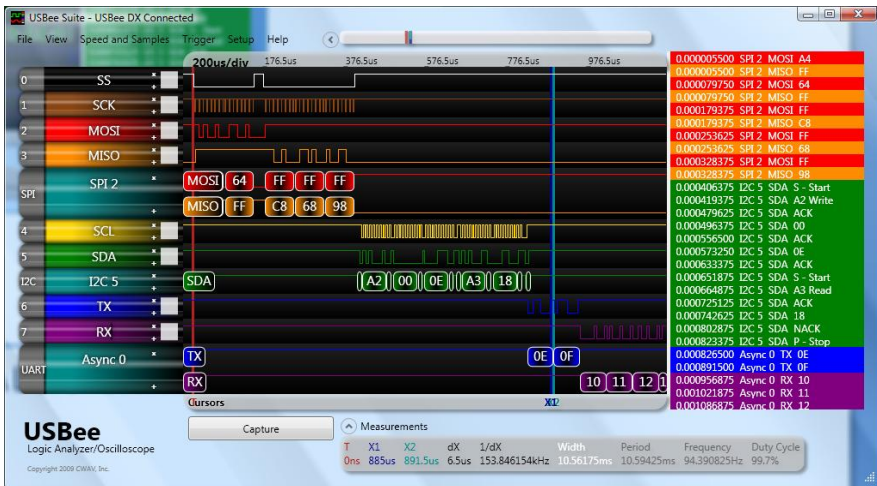
We then add the Async bus using the Channel Settings dialog box as below with the resulting waveline.



Each bus type has various parameters that can be tailored to get the data out of your bus the way you need it.

## DECODED DATA LIST

You can see the decoded list data in vertical format using the View/Show Decode Bus Listing menu item. This opens a window on the right side of the screen that displays the decoded data in vertical format. The data shown is the data that is decoded from the left side of the waveform screen. This data is synchronized to the waveforms as you pan and zoom.

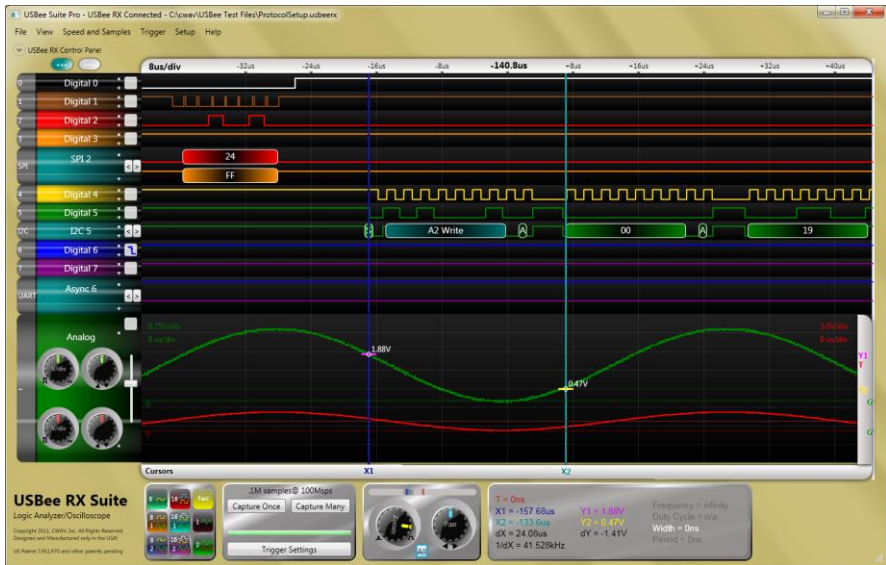


You can hide or change the width of the Decoded Bus Listing window by clicking and dragging the vertical line to the left of the window.

## MANUAL MEASUREMENTS AND CURSORS

The main reason for using an oscilloscope or logic analyzer is to measure the various parts of a waveform. The USBee RX Suite uses cursors to help in these measurements.





The **X1 and X2 Cursors** are placed on any horizontal sample time. This lets you measure the time at a specific location or the time between the two cursors. To place the X cursors, move the mouse to the gray Cursors box just below the waveform. When you move the mouse in this window, you will see a temporary line that indicates where the cursors will be placed. Place the X1 cursor by left clicking the mouse at the current location. Place the X2 cursor by right clicking the mouse at the current location. These cursors will snap to the exact edge of a digital signal when the mouse moves close to the edge. This lets you easily get exact measurements between edges of signals.

In the Measurement window, you will see the various measurements made off of these cursors.

- **X1 Position** – time at the X1 cursor relative to the trigger position
- **X2 Position** – time at the X2 cursor relative to the trigger position
- **RX** – time difference between X1 and X2 cursors
- **1/RX** – the frequency computed using the period between X1 and X2 cursors

## INSTA-MEASUREMENTS

The Insta-Measure feature lets you quickly and accurately measure events and levels by simply hovering the mouse over a signal and without placing cursors.

Insta-Measurements available are as follows:

### Analog Insta-Measurements

- Voltage At Cursor

### Digital Insta-Measurements

- Width, Period, Frequency and Duty Cycle

## BUS DECODING OPTIONS

The USBee RX Suite software has a powerful embedded bus decoder feature that allows you to quickly analyze the contents of embedded communications captured by the pod. This section details each of the available bus types and the parameters required for proper setup.

### GENERIC SIGNAL SETUP

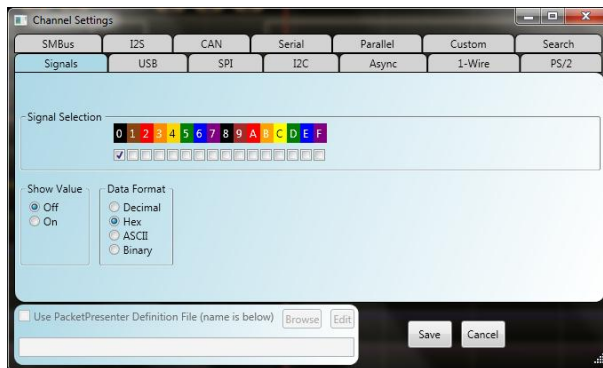
The default trace shown on a waveline is a voltage versus time waveform representation of an input wire on the USBee RX.

#### Hardware Setup

To use the generic signal display you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

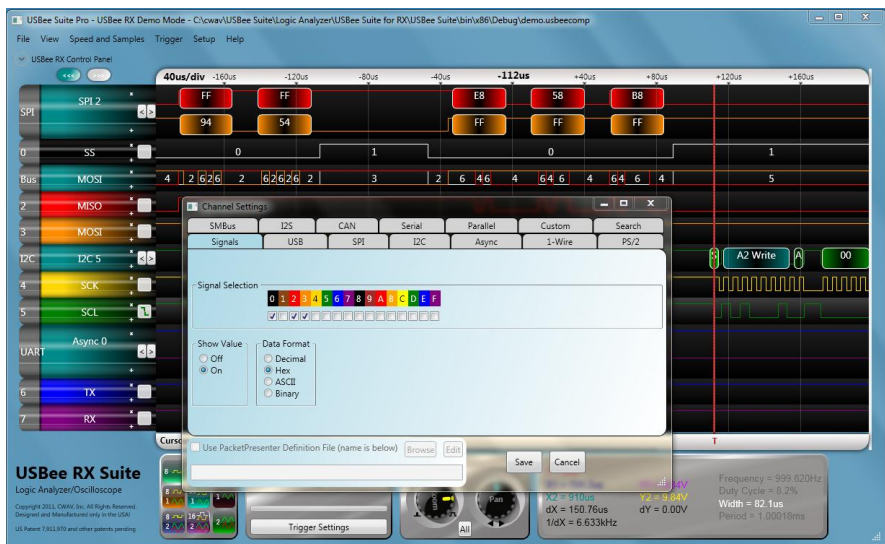
#### Software Setup

Activate the below Channel Settings Dialog by clicking the grey tab on the left of the signal names on the main application screen.



On the above dialog box, select the signals you want to display by checking the associated check box, choose whether to show the value of the selected signals and what format you would like the values.

The signal traffic will be shown as in the following screen. The SS line shows a single signal with the values turned on. The MOSI line shows a combination of three different signals and the resulting value of those signals. The associated settings dialog box is shown.



## CAN BUS SETUP

The CAN Bus Decoder takes the captured data from a CAN bus (11 or 29-bit identifier supported), formats it and allows you to save the data to disk or export it to another application using Cut and Paste.

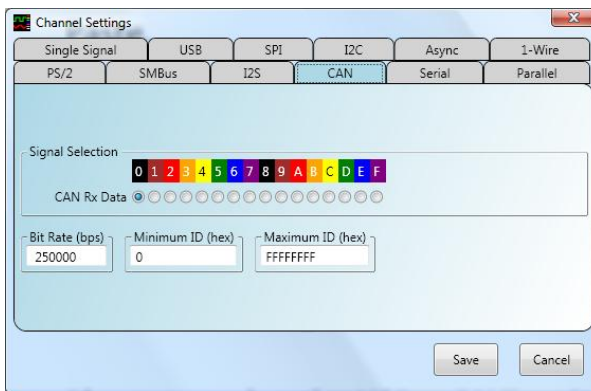
### Hardware Setup

To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The CAN Bus Decoder connects to the digital side of your CAN bus transceiver and only needs to listen to the receiving side of the transceiver (such as the RxD pin on the Microchip MCP2551 CAN bus transceiver chip). Use signal 0 as the RxD data line and connect the GND line to the digital ground of your system. Connect these signals to the CAN bus transceiver IC using the test clips provided.

### Software Setup

Activate the below Channel Settings Dialog by clicking the grey tab on the left of the signal names on the main application screen.



On the above dialog box, select the CAN data signal, what speed the bus is operating at and what filter value for the ID you want (if any)

The bus traffic will be decoded as in the following screen.



## USB BUS SETUP

The USB Bus Decoder decodes Low and Full Speed USB. It does NOT decode High Speed USB. To decode Full Speed USB, the sample rate must be 24Mps, meaning you must sample with just 8 digital channels only. To decode Low Speed USB, you can sample as low as 3Mps.

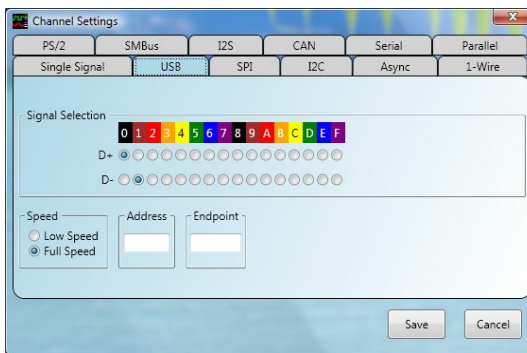
### Hardware Setup

To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Connect two of the digital signals to the D+ and D- of your embedded USB bus, preferably at the IC of the USB device or the connector that the USB cable plugs into.

### Software Setup

Activate the below Channel Settings Dialog by clicking the grey tab on the left of the signal names on the main application screen.



On the above dialog box, select the D+ and D- signals and what speed the bus is operating at. You can also specify a specific USB Address or Endpoint you want to see. All other transactions will be filtered out. Leave the fields blank to see all transactions.

The bus traffic will be decoded as in the following screen.



## I2C BUS SETUP

The I2C Bus Decoder takes the captured data from a I2C bus.

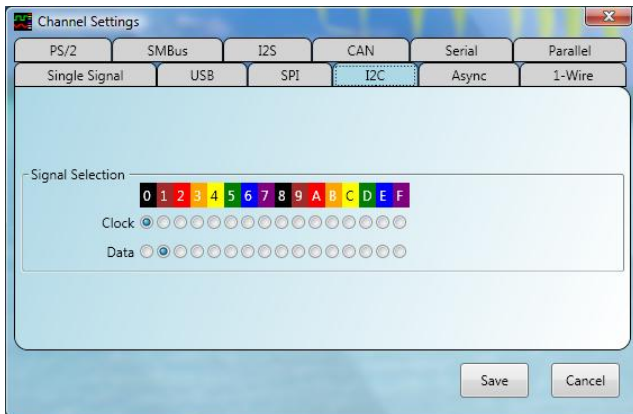
### Hardware Setup

To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The I<sup>2</sup>C Bus Decoder connects to the SDA and SCL lines of the I<sup>2</sup>C bus. Use one signal as the SDA data line and one signal as the SCL clock line. Also connect the GND line to the digital ground of your system. Connect these signals to the I<sup>2</sup>C bus using the test clips provided.

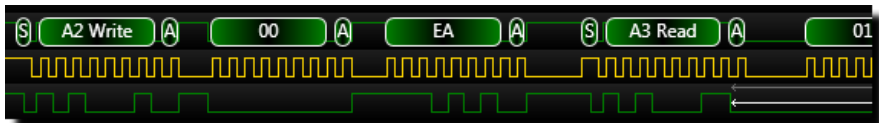
### Software Setup

Activate the below Channel Settings Dialog by clicking the grey tab on the left of the signal names on the main application screen.



On the above dialog box, select the SDA and SCL signals.

The bus traffic will be decoded as in the following screen.



## ASYNCR BUS SETUP

The Async Bus Decoder takes the captured data from an asynchronous bus (UART).

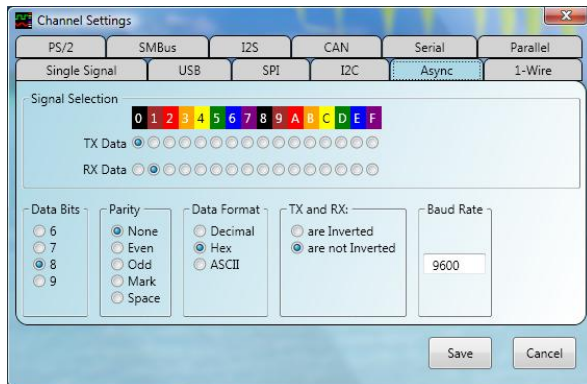
### Hardware Setup

To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The Async Bus Data decoder uses one or more of the 16 digital signal lines (0 thru F) and the GND (ground) line. Connect any of the 16 signal lines to an Async data bus. Connect the GND line to the digital ground of your system.

### Software Setup

Activate the below Channel Settings Dialog by clicking the grey tab on the left of the signal names on the main application screen.



On the above dialog box, select the channels you want to observe. Each channel can be attached to a different async channel. Also enter the baud rate (from 1 to 24000000), the number of data and parity bits, and what output format you want the traffic.

## PARALLEL BUS SETUP

The Parallel Bus Decoder takes the captured data from a parallel bus. The Parallel Bus decoder is also a way to capture the data using an external clock.

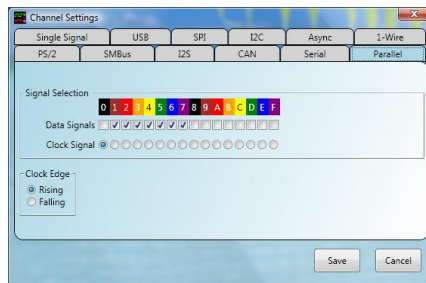
### Hardware Setup

To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The Parallel Bus Data decoder uses the 16 digital signal lines (0 thru F), the GND (ground) line. Connect the GND line to the digital ground of your system.

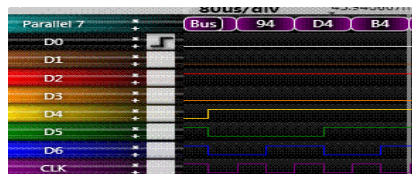
### Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the channels you want to include in the parallel data bus. You can also use any one of the 16 digital signals as an external clock. Choose if you want to use the external clock signal and the external clock edge polarity.

The bus traffic will be decoded as in the following screen.





## 1-WIRE BUS SETUP

The 1-Wire Bus Decoder takes the captured data from a 1-Wire bus.

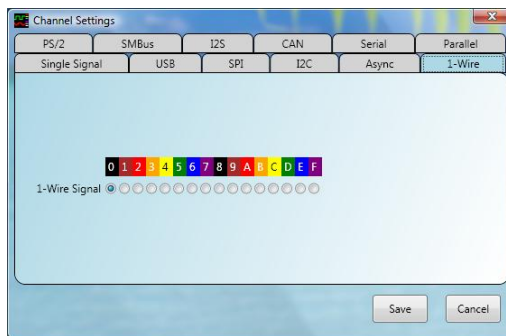
### Hardware Setup

To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The 1-Wire Bus Data decoder uses any one of the 16 digital signal lines (0 thru F), the GND (ground) line. Connect the GND line to the digital ground of your system.

### Software Setup

Activate the below Channel Settings Dialog by clicking the grey tab on the left of the signal names on the main application screen.



On the above dialog box, select the signal running your 1-Wire protocol.

## SPI BUS SETUP

The SPI Bus Decoder takes the captured data from an SPI bus.

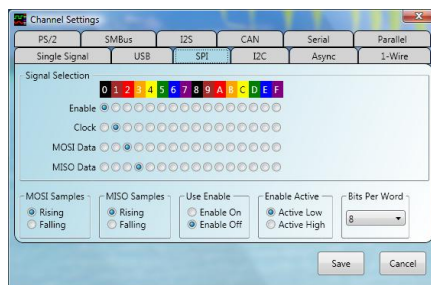
### Hardware Setup

To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The SPI Bus Decoder uses any one of the 16 digital signal lines (0 thru F) for the SS (slave select), SCK (clock), MISO (data in), MOSI (data out), and the GND (ground) line. Connect the SS, SCK, MISO, and MOSI to your digital bus using the test leads and clips. Connect the GND line to the digital ground of your system.

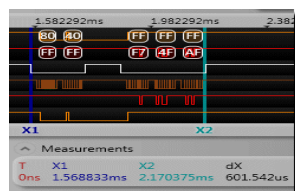
### Software Setup

Activate the below Channel Settings Dialog by clicking the grey tab on the left of the signal names on the main application screen.



On the above dialog box, select the signals you plan to use for the SPI protocol. Also set the appropriate sampling edges for both data lines and if you would like to use the SS (slave select) signal. If you turn off the SS, all clocks are considered valid data bits starting at the first clock detected. Also choose what output format you want the traffic.

The bus traffic will be decoded as in the following screen.



## SDIO BUS SETUP

The SDIO Bus Decoder shows the captured data from an SDIO bus Command and Data lines.

### Hardware Setup

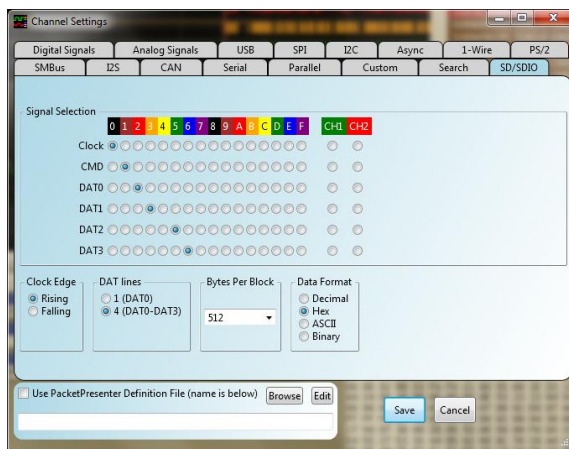
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The SDIO Bus Decoder uses any one of the 16 digital signal lines (0 thru F) for the CMD (command), CLK (clock), DAT0, DAT1, DAT2, DAT3 and the GND (ground) line. Connect the CMD, CLK and DAT0-DAT3 lines to your digital bus using the test leads and clips. Connect the GND line to the digital ground of your system.

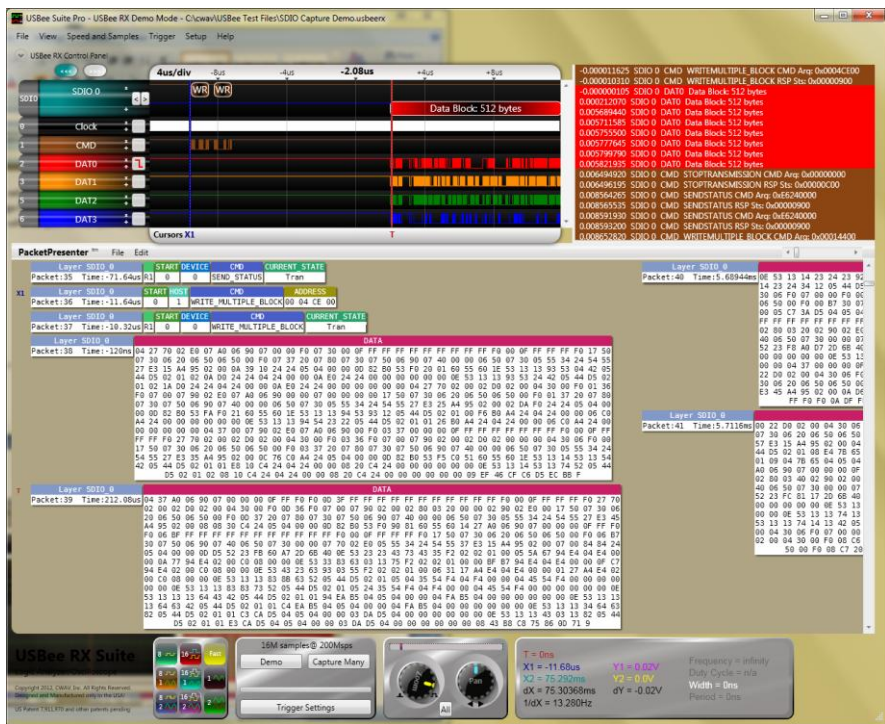
### Software Setup

You will want to make sure that the sample rate you have chosen is at least 4 times the CLK frequency in order to recover the bus data reliably. So, for a CLK with a frequency of 50MHz, you need at least a sample rate of 200Mps. Also remember that at sample rates above 100Mps the available channels is reduced.

Activate the below Channel Settings Dialog by clicking the grey tab on the left of the signal names on the main application screen.



On the above dialog box, select the signals you plan to use for the SDIO protocol. Also set the appropriate sampling edge for the CMD and DAT lines, how many DAT lines to use and the size of the data blocks. Also choose the output format for the traffic.



## SM BUS BUS SETUP

The SM Bus Decoder takes the captured data from an SM bus.

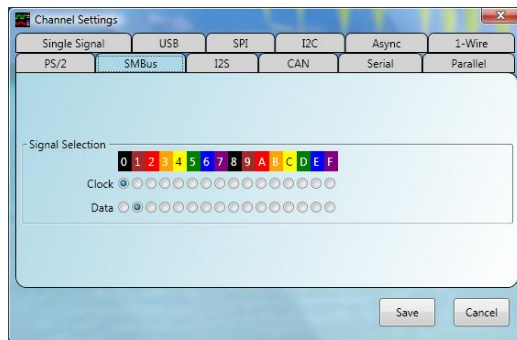
### Hardware Setup

To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The SM Bus Decoder uses any one of the 16 digital signal lines (0 thru F) for the SM Clock and SM Data, and the GND (ground) line. Connect the SM Clock and SM Data to your digital bus using the test leads and clips. Connect the GND line to the digital ground of your system.

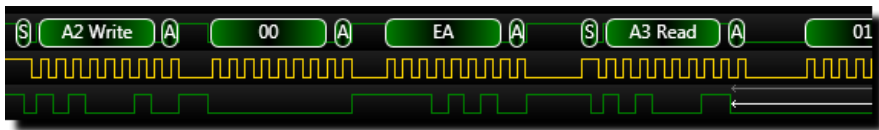
### Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the signals you plan to use for the SM Bus protocol.

The bus traffic will be decoded as in the following screen.



## SERIAL BUS SETUP

The Serial Bus Decoder takes the captured data from a Serial bus. The serial data can be from any clocked serial bus and can be aligned using a hardware signal or an embedded sync word.

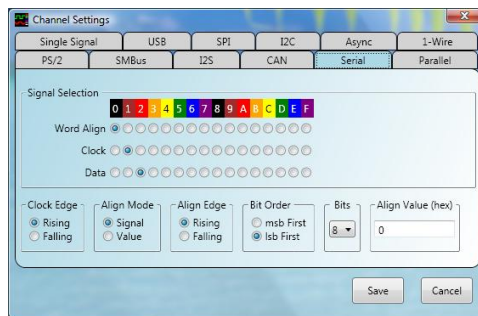
### Hardware Setup

To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The Serial Bus Decoder uses any one of the 16 digital signal lines (0 thru F) for the Clock, Data and optional Word Align signal, and the GND (ground) line. Connect the Clock, Data and Word Align to your digital bus using the test leads and clips. Connect the GND line to the digital ground of your system.

### Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the signals you plan to use for the Serial Bus protocol. Select whether you have an external word align signal (Align Mode = Signal) or if your serial data has an embedded sync word in the data stream (Align Mode = Value). The Bits/Word is the size of the Sync word as well as the output word size. Choose the bit ordering as well as the output format of the traffic.

The bus traffic will be decoded as in the following screen.



## I2S BUS SETUP

The I2S Bus Decoder takes the captured data from an I2S bus.

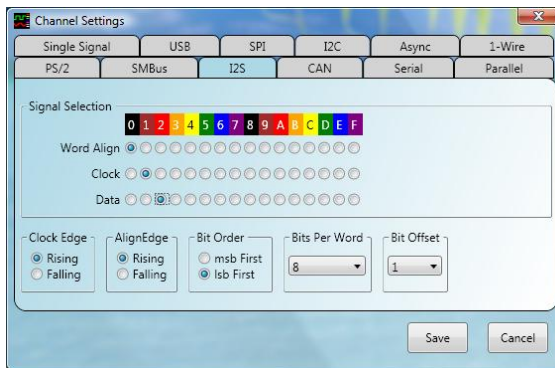
### Hardware Setup

To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The I2S Bus Decoder uses any one of the 16 digital signal lines (0 thru F) for the Clock, Data and Word Align signal, and the GND (ground) line. Connect the Clock, Data and Word Align to your digital bus using the test leads and clips. Connect the GND line to the digital ground of your system.

### Software Setup

Activate the below Channel Settings Dialog by clicking the grey tab on the left of the signal names on the main application screen.



On the above dialog box, select the signals you plan to use for the I2S Bus protocol. Select the start edge for the external word align signal, the Bits/Word and the Clock sampling edge. Choose the bit ordering.

The bus traffic will be decoded as in the following screen.



## PS/2 BUS SETUP

The PS/2 Bus Decoder takes the captured data from an PS/2 bus.

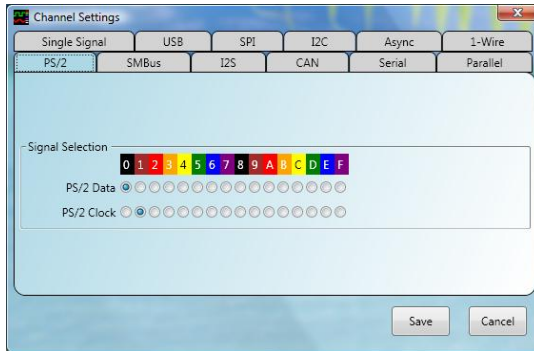
### Hardware Setup

To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The PS/2 Bus Decoder uses any one of the 16 digital signal lines (0 thru F) for the Clock and Data signals, and the GND (ground) line. Connect the Clock and Data to your PS/2 bus using the test leads and clips. Connect the GND line to the digital ground of your system.

### Software Setup

Activate the below Channel Settings Dialog by clicking the grey tab on the left of the signal names on the main application screen.



On the above dialog box, select the signals you plan to use for the PS/2 Bus protocol.

The bus traffic will be decoded as in the following screen.



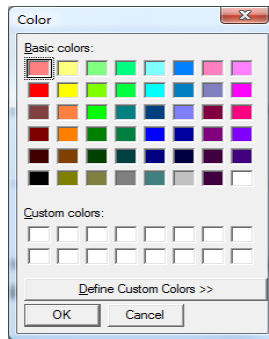


## SETTING VIEWING PREFERENCES

The USBee RX Suite has many ways that you can customize the display of your data, beyond the placement of the waveforms.

### CURSOR COLORS

You can change the color of the Trigger, X1 and X2 cursors using the View Menu. When chosen you will see the Color Selection dialog box below. To change the colors back to their original state, use the View/Reset Colors To Default menu item.

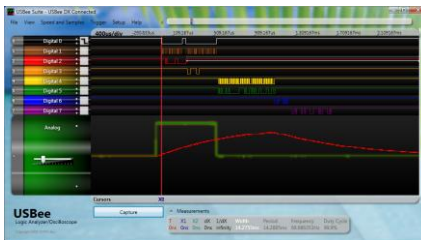


### BACKGROUND COLOR

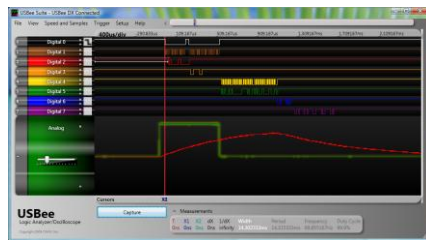
The background of the waveform screen can be set to white or black using the View | Background White or View | Background Black menu items.

### GLASS APPEARANCE

Windows has added the ability to have a Glassy appearance on applications. If your system is capable of this glassy look you can use the View/Show Glass If Possible menu item to turn it on. If it is not possible, or you turn off Glass, the display shows a grey background.



With Glass On



With Glass Off

## FILE OPERATIONS

Using the File menu you can start a New file, Save trace data, Open previously saved traces and Export trace data to other file formats.

---

### CREATING A NEW FILE

To start a new file, choose File/New. This will configure the screen to the default state with all available channels enabled.

---

### SAVING A CAPTURE FILE

After capturing a trace, you can save it to disk using the File/Save As menu item. This saves all trace data, cursor positions and screen format. The files can be saved in either Uncompressed format (.usbeesuite extension) or in Compressed format (.usbeecomp extension) to reduce drive space requirements. Saving and Opening large buffer sizes can take a while to perform the compression and saving but can greatly reduce disk space.

---

### OPEN AND EXISTING CAPTURE FILE

To view a previously saved capture file, use the File/Open menu item. This will load the trace data and screen format including decoder setup. Opening files with large buffer sizes can take a while to decompress and display.

---

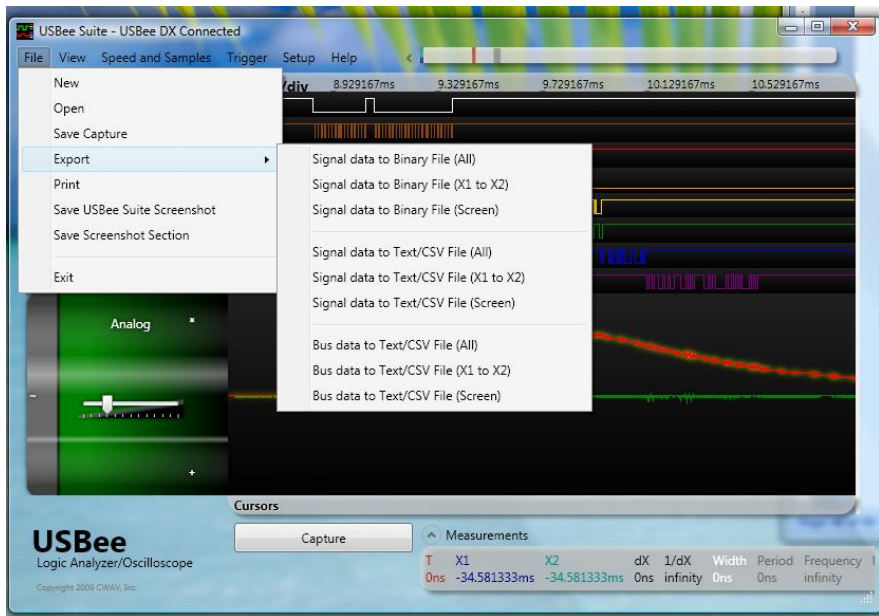
### RECENTLY USED FILE LIST

The USBee RX Suite maintains a recently used file list that allows you to quickly load any of the last 5 previously used files. Simply click on the file name in the File, Recently Used file list to open it.

---

### EXPORTING CAPTURED DATA TO A FILE

Since the compressed trace files are not in easily useable format, you can use the File/Export menu items to save the trace and decoded data into formats that are easy to use.



The available options for exporting are:

- Save the Signal Data to a Binary File
- Save the Signal Data to a Text/CSV File
- Save the Bus Data to a Text/CSV file

You can specify the range of samples to export by using the All, X1 to X2, or Screen versions. Screen will output all samples viewed on the current screen, X1 to X2 will output all samples between the X1 and X2 cursor, and All will output all samples in the sample buffer. Choosing All will create VERY large files, so use with caution.

## EXPORT SIGNAL DATA TO BINARY FILE

When exporting signal data to a binary file, each sample is made up of 4 bytes. Each sample was taken at the sample rate that was set at the time of capture. A single sample (4 bytes) is formatted as follows:

Byte	Description
1	Digital channels 0 to 7 (lsb= signal 0, msb = signal 7)
2	Digital channels 8 to F (lsb= signal 8, msb = signal F)
3	Channel 1 Analog voltage (0=-6V, 128 = 0V, 255 = +6V)
4	Channel 2 Analog voltage (0=-6V, 128 = 0V, 255 = +6V)

---

## EXPORT SIGNAL DATA TO TEXT/CSV FILE

When exporting signal data to a text/csv file, each sample is output to a single line with each signal separated by a comma. Each sample was taken at the sample rate that was set at the time of capture. An example output file is formatted as follows showing a header that specifies the column labels and which signal is associated:

```
Time,0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
0.008739333,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008739500,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008739667,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008739833,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008740000,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008740167,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008740333,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008740500,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008740667,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008740833,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
```

---

## EXPORT BUS DATA TO TEXT/CSV FILE

When exporting bus data to a text/csv file, each decoded element is output to a single line with each field separated by a comma. An example output file is formatted as follows showing a header that specifies the column labels and which signal is associated:

```
Time(seconds),Bus Name,Signal Name,Data
0.000007167,SPI 3,MOSI,FF
0.000007167,SPI 3,MISO,24
0.000106333,SPI 3,MOSI,FF
0.000106333,SPI 3,MISO,A4
0.000238667,SPI 3,MOSI,48
0.000238667,SPI 3,MISO,FF
0.000338000,SPI 3,MOSI,A8
0.000338000,SPI 3,MISO,FF
0.000437167,SPI 3,MOSI,18
0.000437167,SPI 3,MISO,FF
0.000542667,I2C 5,SDA,S - Start
0.000558500,I2C 5,SDA,A2 Write
0.000640333,I2C 5,SDA,ACK
0.000661167,I2C 5,SDA,00
0.000743000,I2C 5,SDA,ACK
0.000763667,I2C 5,SDA,0D
0.000845500,I2C 5,SDA,ACK
0.001098833,I2C 5,SDA,P - Stop
```

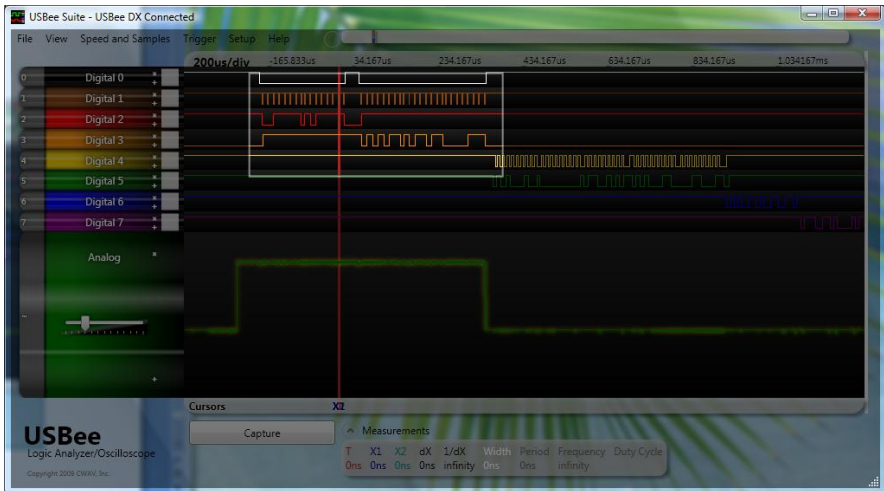
## PRINTING

To print an image of the current screen, choose File/Print from the menu.

## CREATING SCREEN SHOTS

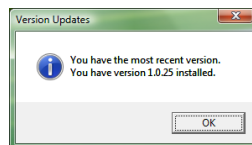
An easy way to create documentation is to take screen shots, or portions of the screen image, and save them to graphics files. You can save the entire USBee RX Suite application window to a file using the File/Save USBee RX Suite Screenshot menu item. This lets you save the image as a BMP, JPG, PNG, GIF, TIF, or WMF file to be used by your favorite graphics program.

You can use the File/Save Screenshot Section menu item to select just a portion of the screen to save. Use the left mouse button to start a rectangle that selects the region to save. When you let up on the button it will prompt you for the filename to save the image as.



## SOFTWARE UPDATES

New versions of the USBee RX Suite software are posted on the USBee.com web site. To have the USBee RX Suite software check if a new version exists, use the Help/Check for Updates menu item. It will connect to the USBee.com server and determine if there is a newer version available for download. If you are up to date, the following screen will appear.



## DEVELOPING YOUR OWN CUSTOM DECODERS

The USBee RX Suite allows you to create your own custom protocol decoders.

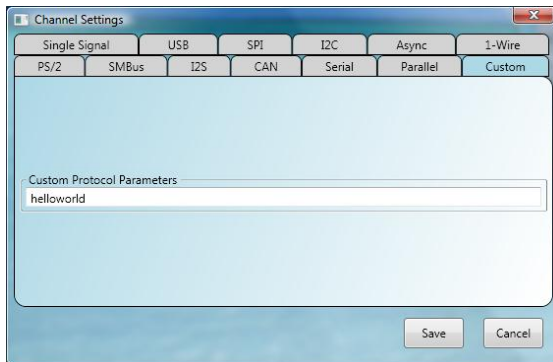
To implement a custom decoder you must create a Class Library (DLL) using the code below as an example. You can build this Class Library using the free Microsoft Visual Studio 2008 Express or newer. Our example is in Visual Basic, but can easily be ported to C or other language supported in Visual Studio.

We will first show how to use a Custom Decoder and then show how to design one.

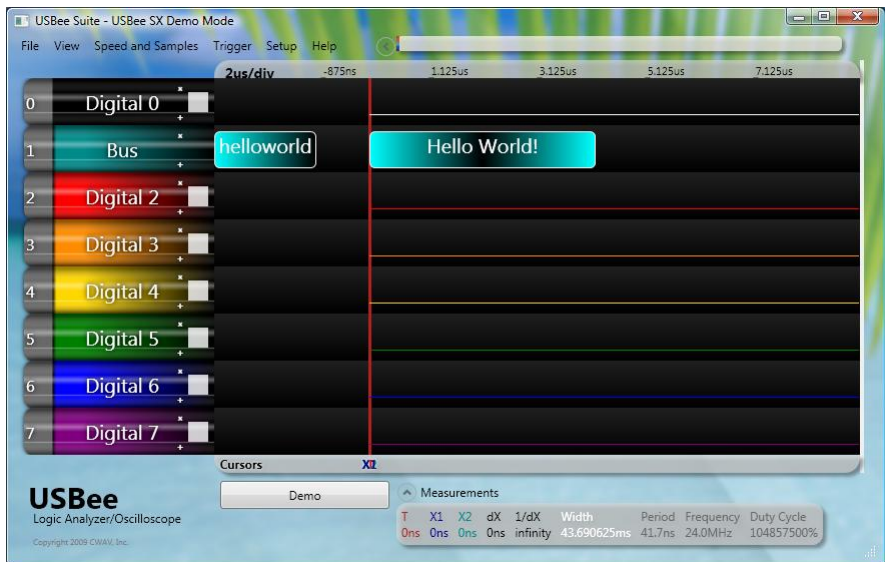
### USING THE CUSTOM DECODER

Using the Custom Decoder that you build is simple. To select to use your Custom Decoder you select the Custom tab in the Channel Settings dialog box. You can then enter a set of parameters that are sent to your decoder. These parameters can specify anything you may need to determine how to decode your protocol, including which protocol, which signals to use, baud rates, inversions, etc. and is purely defined by you.

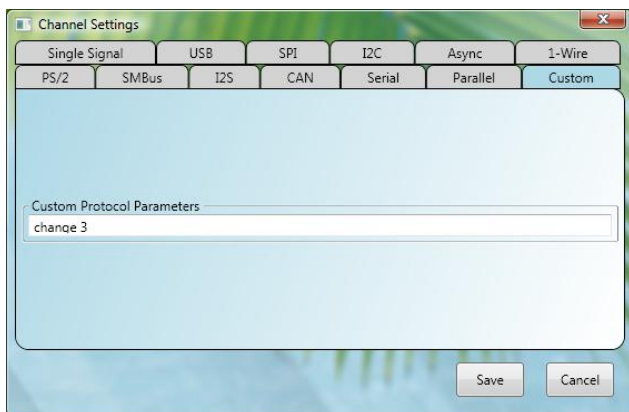
Below is the Channel Setting screen showing that we want to use our Custom Decoder on this waveline. We are also passing the text string “helloworld” to the decoder when it runs.



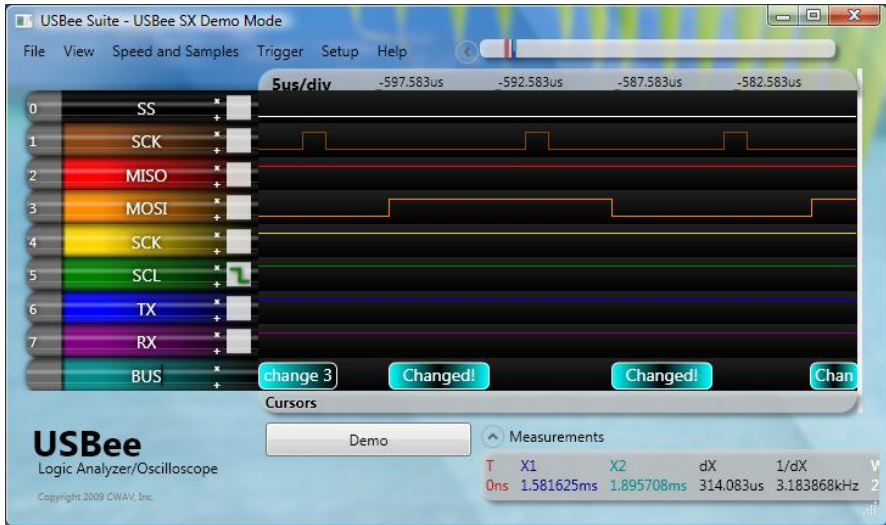
When we press Save our Decoder is run, passing the parameters to it and the resulting Entries are displayed. Below is the output from our VERY simple Hello World decoder which puts a “Hello World!” at the first sample.



On a different Waveline you can specify a different set of Parameters (in this case “change 3”) that indicate to the Custom Decoder to perform an entirely different decode.



As in our example decoder, this “Change” indicates to place an Entry at each change of state of the specified signal (in this case 3). The resulting display is as follows.



Obviously much more complicated protocols can be decoded using these simple methods of parameter passing and Entry displaying.

## BUILDING THE CUSTOM DECODER

To implement a custom decoder you must create a Class Library (CustomUSBeeSuiteDecoder.DLL) using the code below as an example. This example code is also included when you install the USBee RX Suite software in the \Program Files\CWAV Inc\USBee RX Suite\CustomUSBeeSuiteDecoderRX\ directory. You can build this Class Library using the free Microsoft Visual Studio 2010 Express. Our example is in Visual Basic, but can easily be ported to C or other language supported in VS2010.

The main function of a Custom Decoder code is below.

1. Receive parameters for the protocol from the User Interface
2. Access the sample data and decode the protocol based on the parameters
3. Output "Entries" that consist of a Start Sample, End Sample and a Text String

Once you create your own CustomUSBeeSuiteDecoderRX.DLL file, you simply copy your new file over the one that was provided with the original install in the \Program Files\CWAV Inc\USBee RX Suite directory. You may need to locate this file on your system and have administrator rights in order to replace it.



## EXAMPLE CLASS LIBRARY CODE

Below is our example Class Library source code that performs a few different protocol decodes and displays the results on the waveline. A version that includes an actual NEC IR decoder is installed with the USBees RX Suite. Use this example to start your own.

```
Option Explicit On
Option Strict On
Imports System.IO

Public Class CustomUSBeesSuiteDecoderRX

    Declare Function SampleData Lib "usbeexste.dll" Alias "?LoggedData@YGJ_U8Z" (ByVal Index As Int64) As Integer
    Declare Function FindNextEdge Lib "usbeexste.dll" Alias "?FindNextEdge@YG_UJKRZ" (ByVal start As Int64, ByVal Mask As Integer,
    ByVal direction As Integer) As Int64

    ' The SampleData routine returns a 4 byte value that contains a single sample of all the signals
    ' The format of the 32 bits returned is as follows:
    '
    '
    '      MSB                                LSB
    ' XXXXXXXXXXXXXXXXXXFFEDCBA9876543210
    '
    ' where XXXXXXXX is Channel 2 Analog value (0=-6V, 255 = +6V)
    ' YYYYYYYY is Channel 1 Analog value (0=-6V, 255 = +6V)
    ' F is logic level (0 or 1) for channel F
    ' E is logic level (0 or 1) for channel E
    ' D is logic level (0 or 1) for channel D
    ' ..
    ' 0 is logic level (0 or 1) for channel 0

    ' FindNextEdge routine scans the sample buffer starting at the start sample number to find the next sample
    ' with an edge on any masked channels. Mask bits aligned as above with 1 considered, 0 being ignored.
    ' direction 1 = forward, 0 = backward.
    ' FindNextEdge should be used instead of SampleData whenever possible since it is much faster.
    ' Return value is the sample number at the next edge or -1 if there is no more edges in that direction.

    Dim GTriggerSample As Int64
    Dim GX1Sample As Int64
    Dim GX2Sample As Int64

    Public Sub SetCaptureParameters(ByVal TriggerSample As Int64, ByVal X1Sample As Int64, ByVal X2Sample As _ Int64)
        ' This routine is called at the end of a capture to pass the Trigger and X1 and X2 Cursors to the
        ' custom decoding process.
        GTriggerSample = TriggerSample
        GX1Sample = X1Sample
        GX2Sample = X2Sample
    End Sub

    Public Sub DecodeCustom(ByVal OutFilename As String, ByVal NumberOfSamples As Int64, ByVal SamplingRate As _ Integer, ByVal Parameters
    As String)
        Dim OldSample As UIntInteger

        Try
            ' This is a custom bus decoder Processing Routine
            '
            ' The passed in variables are as follows:
            ' OutFilename - the file that all of the decoded Entries get written to. This is the file
            ' that the USBees Suite
            ' will read to display the data on the waveline.
            ' ActualNumberOfSamples - How many samples are in the sample buffer
            ' SamplingRate - The rate that the samples were taken in sps. 24000000 = 24Mpsps...
            ' Parameters - User defined string passed from the USBees Suite user interface Channel Setting
            ' for the custom decoder.
            ' Use this string to pass in any parameters that your decoder needs to know,
            ' such as what channels to use
            ' in decoding, which protocol if you have multiple protocols supported here, and
            ' how you want the data formatted.

            ' Below is an example set of Custom Protocol decoders that show how to access the sample buffer
            ' and how to generate output that get sent to the screen.

            ' Setup the File Stream that stores the Output Entry Information
            Dim FS As New FileStream(OutFilename, FileMode.Append, FileAccess.Write)
            Dim BW As New BinaryWriter(FS)
            Dim Sample As Int64

            SampleData(0) ' Initializes the sample retrieval

            ' Since this file supports different custom decoders, we need to see which one to run based on the Parameters string
            If CBool(InStr(Parameters.ToUpper, "CHANGE")) Then
                ' Sample Decoder that just detects when a signal changes state
                ' The signal to use for the detection is specified in the Parameters as the second parameter

                Dim Params() = Parameters.Split(CChar("-"))
                Dim SignalToUse As Double = Val(Params(1))
                Dim SignalMask As Integer = 1 << CInt(SignalToUse) ' Make the mask that will mask off the channel we want in the sample

                ' Now go from the start of the samples to the end and process the decoder
                For Sample = 0 To NumberOfSamples - 1
                    ' This example decoder places a label at every transition of a digital signal
                    Dim NextEdge = FindNextEdge(Sample, SignalMask, 1)
                    If NextEdge >= 0 Then ' Edge was found
                        WriteEntry(BW, NextEdge, NextEdge + 100, "Changed! S:" & Sample & "H:" & NextEdge)
                    Else
                        ' There are no more edges so end
                        Exit For
                    End If
                    Sample = NextEdge - 1
                Next

            ElseIf CBool(InStr(Parameters.ToUpper, "RISE")) Then
                ' Sample Decoder that just detects when a signal changes state
                ' The signal to use for the detection is specified in the Parameters as the second parameter
            End If
        Catch ex As Exception
            ' Handle exception
        End Try
    End Sub
End Class
```

```

Dim Params() = Parameters.Split(CChar("-"))
Dim SignalToUse As Double = Val(Params(1))
Dim SignalMask As Integer = 1 << CInt(SignalToUse) ' Make the mask that will mask off the channel we want in the sample

' Now go from the start of the samples to the end and process the decoder
For Sample = 0 To NumberOfSamples - 1
    ' This example decoder places a label at every transition of digital signal 0
    Dim DigitalChannel As Integer = CInt(SampleData(Sample) And SignalMask)
    If (DigitalChannel <> OldSample) And (OldSample = 0) Then
        WriteEntry(BW, Sample, Sample + 100, "Rise!")
    End If
    OldSample = DigitalChannel
Next

ElseIf CBool(InStr(Parameters.ToUpper, "HELLOWORLD")) Then
    ' Simplest Decoder Possible
    ' Print Hello World at the start of the buffer
    WriteEntry(BW, 0, 100, "Hello World!")
    WriteEntry(BW, GTriggerSample, GTriggerSample + 100, "Trigger Is Here!")
    WriteEntry(BW, GX1Sample, GX1Sample + 100, "X1 Is Here!")
    WriteEntry(BW, GX2Sample, GX2Sample + 100, "X2 Is Here!")

ElseIf CBool(InStr(Parameters.ToUpper, "NECIR")) Then
    ' Sample Decoder that just detects when a signal changes state
    ' The signal to use for the detection is specified in the Parameters as the second parameter
    WriteEntry(BW, 0, 100, "NEC IR Decoder 3.0")

Dim Params() = Parameters.Split(CChar("-"))
Dim SignalToUse As Double = Val(Params(1))
Dim SignalMask As Integer = 1 << CInt(SignalToUse) 'Make the mask that will mask off the channel we want in the sample

Const LOOKING_FOR_HEADER As Integer = 1
Const LOOKING_FOR_BITS As Integer = 2
Dim DecodeState As Integer = LOOKING_FOR_HEADER ' Holds what state of the decoder we are in

Dim ByteAccumulator As Integer = 0 ' Holds the accumulated bits for each byte
Dim BitCounter As Integer = 0 ' Holds how many bits we have accumulated in this byte so far
Dim ByteStartSample As Int64 ' Holds the sample at the start of the byte

Dim Data As UInteger ' Holds the state of the signal at the current sample
Dim tEdge1 As Int64 ' Where the first edge is
Dim tEdge2 As Int64 ' Where the second edge is
Dim tPulseWidth As Double ' The pulsewidth in seconds

' Now go from the start of the samples to the end and process the signal
For Sample = 0 To NumberOfSamples - 1
    Data = CUInt(SampleData(Sample) And SignalMask)
    If DecodeState = LOOKING_FOR_HEADER Then
        If Data <> 0 Then
            ' We found a High which starts the Header
            ' Now look for the next edge
            tEdge1 = FindNextEdge(Sample, SignalMask, 1)
            If tEdge1 >= 0 Then
                ' Check to see if this falling edge is in the right timeframe
                tPulseWidth = (tEdge1 - Sample) / SamplingRate
                If (tPulseWidth >= 0.008) And (tPulseWidth <= 0.01) Then
                    ' Now look for the rising edge
                    tEdge2 = FindNextEdge(tEdge1, SignalMask, 1)
                    If tEdge2 >= 0 Then
                        ' Check to see if this rising edge is in the right timeframe
                        tPulseWidth = (tEdge2 - tEdge1) / SamplingRate
                        If (tPulseWidth >= 0.004) And (tPulseWidth <= 0.005) Then
                            ' Great! Valid Header Format! Look for bits from this point on
                            ' Write out a Header Marker (remove this if you don't need the header)
                            WriteEntry(BW, Sample, tEdge2, "Lead Code")
                            DecodeState = LOOKING_FOR_BITS
                            Sample = tEdge2
                            ' Initialize the Byte Accumulation variables
                            ByteStartSample = Sample
                            ByteAccumulator = 0
                            BitCounter = 0
                            Continue For
                        ElseIf (tPulseWidth >= 0.002) And (tPulseWidth <= 0.003) Then
                            ' Great! Valid Repeat Format!
                            ' Find the next falling edge to complete the Repeat
                            tEdge2 = FindNextEdge(tEdge2, SignalMask, 1)
                            ' Write out a Repeat Marker (remove this if you don't need the Repeat)
                            WriteEntry(BW, Sample, tEdge2, "Repeat")
                            Sample = tEdge2
                            Continue For
                        Else
                            ' Pulse is not the right size so bail and keep looking
                            Sample = tEdge2
                            Continue For
                        End If
                    Else
                        ' No edges at all! So we are done
                        Exit For
                    End If
                Else
                    ' Pulse is not the right size so bail and keep looking
                    Sample = tEdge1
                    Continue For
                End If
            Else
                ' No edges at all! So we are done
                Exit For
            End If
        End If
    End If
End For

```

```

End If

ElseIf DecodeState = LOOKING_FOR_BITS Then

If BitCounter = 8 Then
' We have an entire byte worth of data so output the information
WriteEntry(BW, ByteStartSample, tEdge2, Hex(ByteAccumulator))
BitCounter = 0
ByteAccumulator = 0
End If

If Data <> 0 Then
' We found a High which starts the bit
' Now look for the next edge

tEdge1 = FindNextEdge(Sample, SignalMask, 1)

If tEdge1 >= 0 Then

' Check to see if this falling edge is in the right timeframe
tPulseWidth = (tEdge1 - Sample) / SamplingRate

If (tPulseWidth >= 0.0005) And (tPulseWidth <= 0.0007) Then
' Good start of a bit
' Now look for the rising edge

tEdge2 = FindNextEdge(tEdge1, SignalMask, 1)

If tEdge2 >= 0 Then
' Check to see if this rising edge is in the right timeframe for a logic "0"
tPulseWidth = (tEdge2 - tEdge1) / SamplingRate
If (tPulseWidth >= 0.0004) And (tPulseWidth <= 0.0006) Then
' Great! Valid 0 Bit Format!

' Write out a Bit Marker (remove this if you don't need the bit)
'WriteEntry(BW, CUInt(Sample), CUInt(tEdge2), "0")

' Add this bit to the accumulators (LSB first)
ByteAccumulator = ByteAccumulator >> 1 ' Shift the Accumulator
ByteAccumulator = ByteAccumulator And &H7F ' Clear out the MSBit

' Mark the start of the byte if so
If BitCounter = 0 Then ByteStartSample = Sample

' Next Bit next time
BitCounter = BitCounter + 1

Sample = tEdge2
Continue For
ElseIf (tPulseWidth >= 0.001) And (tPulseWidth <= 0.002) Then
' Great! Valid 1 Bit Format!

' Write out a Bit Marker (remove this if you don't need the bit)
'WriteEntry(BW, CUInt(Sample), CUInt(tEdge2), "1")

' Add this bit to the accumulators (LSB first)
ByteAccumulator = ByteAccumulator >> 1 ' Shift the Accumulator
ByteAccumulator = ByteAccumulator Or &H80 ' Set the MSBit

' Mark the start of the byte if so
If BitCounter = 0 Then ByteStartSample = Sample

' Next Bit next time
BitCounter = BitCounter + 1

Sample = tEdge2
Continue For

Else
' Pulse is not the right size so bail and keep looking
DecodeState = LOOKING_FOR_HEADER
Sample = tEdge2
Continue For
End If
Else
' No edges at all! So we are done
Exit For
End If

Else
' Pulse is not the right size so bail and keep looking
Sample = tEdge1
DecodeState = LOOKING_FOR_HEADER
Continue For
End If
Else
' No edges at all! So we are done
Exit For
End If

End If

End If

Next

End If

' Close the Output File
FS.Close()

Catch ex As Exception

End Try

End Sub

Public Sub WriteEntry(ByRef BW As BinaryWriter, ByVal StartSample As Int64, ByVal EndSample As Int64, ByRef TextString As String)

' DO NOT CHANGE THIS ROUTINE!!!
' This routine writes the Entry in the file format that is used by the Custom Decoder
' This entry specifies the Start Sample, End Sample and the text string to display
Try

BW.Write(StartSample)
BW.Write(EndSample)

' Write the length of the string in bytes (include the 0 at the end in the count)
Dim tStrLen As UInt32
tStrLen = CUInt(TextString.Length + 1)
BW.Write(tStrLen)


```

```

' Now write out the characters one byte at a time and put a 0 at the end
For x As Integer = 0 To CInt(StrLen - 2)
    BW.Write(CByte(Asc(TextString.Chars(x))))
Next
BW.Write(CByte(0))

Catch ex As Exception

End Try

End Sub

End Class

```

## CUSTOM DECODER PARAMETERS

As you can see in the above code, the Parameter string that is passed from the User Interface to your decoder can be used for a number of purposes. First, it can specify which decoder to run. If you have more than one protocol that you want your decoder to handle, you can select which decoder runs using this text string. For example, in our example Custom Decoder we have a few decoders possible, “CHANGE”, RISE”, “HELLOWORLD” and “NECIR”. Each decoder processes the data differently and outputs different results based on the algorithms.

You can also supply additional parameters in the text string as well as we do in the CHANGE and RISE decoders. The additional parameter in these examples indicates which signal to use to decode.

Again, the definition and use of the Parameter string is entirely up to you but provides an easy to use and simple to implement way to control the behavior of your decoder.

---

## ACCESSING SAMPLE DATA TO PERFORM DECODE

To access each individual sample stored in the sample buffer you use the SampleData or FindNextEdge calls as shown above. This returns a 32-bit value that includes all of the channels levels at that sample time. The format of the 32 bits is as follows:

```
MSB                                     LSB
XXXXXXXXXXXXXXXXXXXXFEDCBA9876543210

where XXXXXXXX is Channel 2 Analog value (0=-10V, 255 = +10V)
YYYYYYYY is Channel 1 Analog value (0=-10V, 255 = +10V)
F is logic level (0 or 1) for channel F
E is logic level (0 or 1) for channel E
...
0 is logic level (0 or 1) for channel 0
```

Decoding any given protocol then entails going through the samples from beginning to end and masking off the channels you need to decode, accumulating decoded bits/bytes along the way and determining what the result is that you want to display.

Since the USBee RX uses sample compression, there may be trillions of actual samples. Therefore, it is best, and will result in faster decoder times, to use the FindNextEdge routine instead of individual SampleData calls when processing a protocol. This routine understands the compression algorithm and does not waste time processing compressed samples.

---

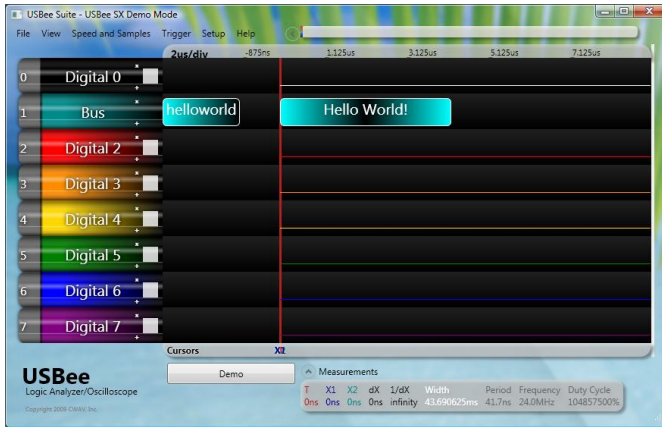
## OUTPUTTING ENTRIES THAT WILL GET DISPLAYED ON THE SCREEN

Once your decoder has determined the result of the decode that you want displayed on the screen, you call the WriteEntry routine above. This gets passed the Start Sample, End Sample and Text String. Entries appear as rounded rectangles on the screen on the associated wave line and are locked to the samples that you specify.

For example, the following call places the “Hello World!” string on the screen stretching from the first sample to the 100<sup>th</sup> sample.

```
WriteEntry(OutFile, 0, 100, "Hello World!")
```

The output is as follows:



## CHANGING THE BACKGROUND COLOR OF OUTPUTTED ENTRIES

You can control the background color of the entire WriteEntry item by embedding a color code anywhere into the text string. Color codes are in the following format:

### [RGB]

Where:

R is the Red value and is a single digit of 0 thru 9.

G is the Green value and is a single digit of 0 thru 9.

B is the Blue value and is a single digit of 0 thru 9.

For example:

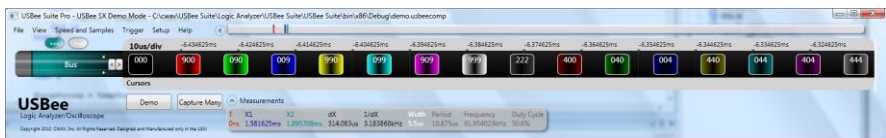
To output an Entry that has a bright green background, use the following:

```
WriteEntry(OutFile, 0, 100, "Hello World![090]")
```

To output an Entry that has a dark red background, use the following:

```
WriteEntry(OutFile, 0, 100, "Hello World![400]")
```

Some common color codes are as follows:



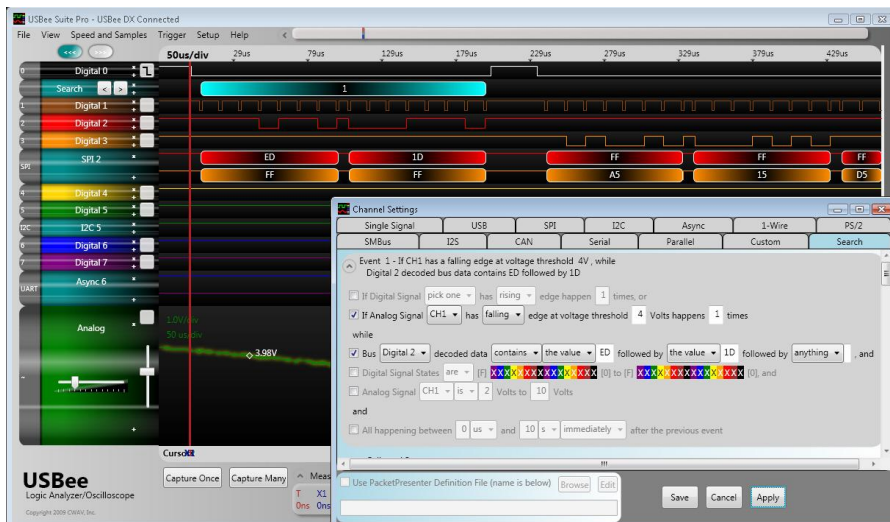
If you do not embed a color code, the background color will be a light cyan.

## SMART SEARCH

The USBee RX Suite Smart Search highlights the sections of your trace matching your areas of interest so that you don't need to waste time hunting for the data you need.

You can specify up to 32 levels of search events that are any combination of bus decoded traffic, states or edges of digital or analog signals, inside or outside of analog voltage ranges and/or digital ranges, and all validated by time specific windows.

Once specified you can pan through the occurrences of your searched items with the click of the mouse and see the total number of times the searched events occur.



## ADDING A SEARCH LINE

To add a Smart Search line on the display, click on the grey tab to the left of the line you want to change (or click the "+" sign to add a new wave line). Once you click on the **Search Tab** you will see the Channel Settings dialog box as below.





## ENTERING A SMART SEARCH

The Smart Search uses the specification that you choose in the channel settings Search window to determine what to highlight on the waveform display.

The search can be made up of up to 32 consecutive **Events**. Each Event is a selection of various bus, analog, and/or digital states. To edit an Event, click on the expander to show the Event details as below.

The screenshot shows the 'Channel Settings' window with the 'Search' tab selected. Under 'Event 1', several search criteria are configured:

- ☐ If Digital Signal **pick one** has **rising** edge happen **1** times, or
- ☐ If Analog Signal **CH1** has **rising** edge at voltage threshold **2** Volts happens **1** times
- while**
- ☐ Bus **pick one** decoded data is **anything** **00** and , and
- ☐ Digital Signal States **are** **[F]** **XXXXXXXXXXXX** **[0]** to **[F]** **XXXXXXXXXXXX** **[0]**, and
- ☐ Analog Signal **CH1** **is** **-10** Volts to **10** Volts
- and**
- ☐ This Event happening between **0** **us** and **10** **s** **immediately** after the previous event

Below the criteria is a 'Followed By' section with a dropdown menu showing 'Event 2 -'. At the bottom, there is a checkbox for 'Use PacketPresenter Definition File (name is below)' with 'Browse' and 'Edit' buttons, and 'Save', 'Cancel', and 'Apply' buttons.

To enable a line in the Event search, click on the checkbox to the left of the items you want to include in your search.

You can specify any combination of the available lines in a search Event. All search criteria in a single Event must occur simultaneously for an event to be considered a match. For example, if you are looking for an edge on a digital signal, and looking for an analog voltage range, the edge must occur **while** the voltage is inside the range to be considered a match.

The following sections detail the available search items for each event and how they function.

## DIGITAL SIGNAL EDGES

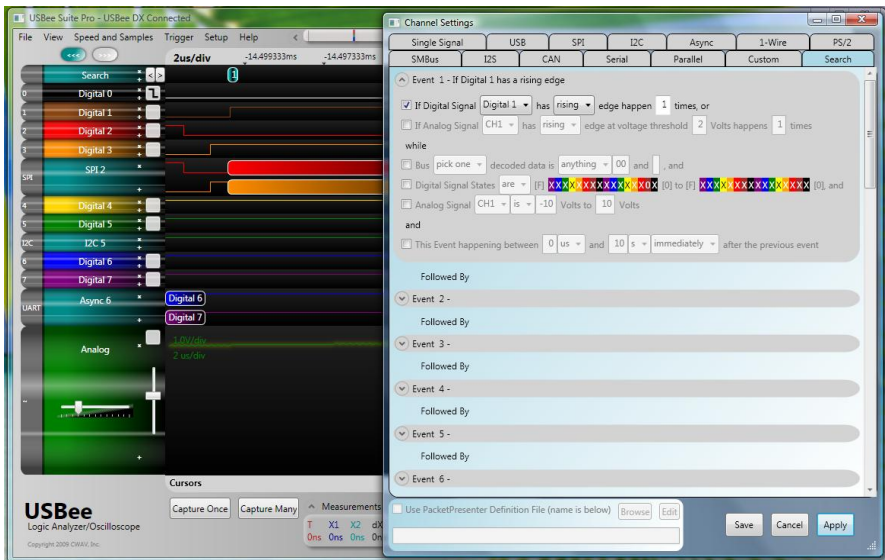
The first line lets you find edges on the digital signals.

First select the Digital line you want to search using the first dropdown box. This box is filled with only the single signals that are displayed on the screen.

Then choose the edge you want to search for: rising, falling or either rising or falling.

Finally, choose how many edges you need to find consecutively. If you want to find areas that have NO edges you can specify 0.

Below shows a search that finds all rising edges of the signal Digital 1.



## ANALOG SIGNAL EDGES

The second line lets you find edges on the analog signals, if your USBee has them.

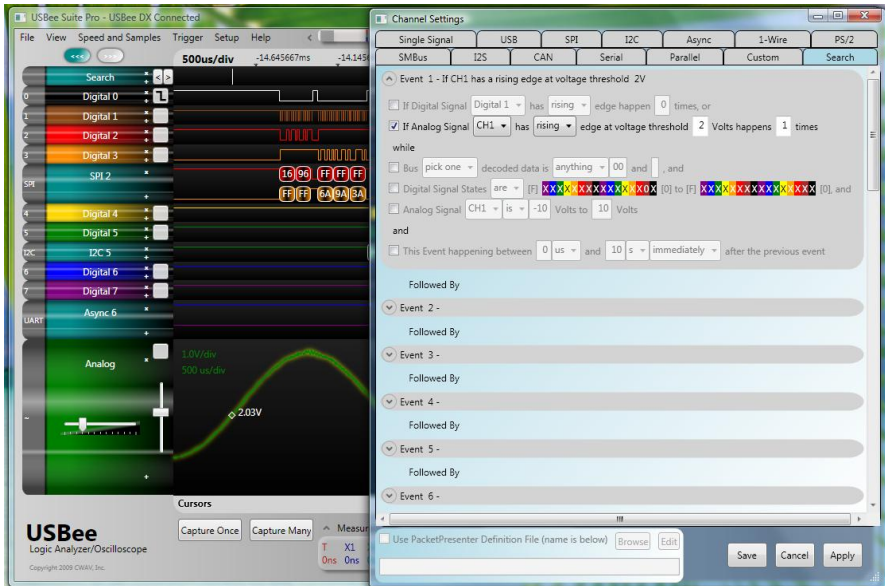
First select the Analog channel you want to search using the first dropdown box.

Then choose the edge you want to search for: rising, falling or either rising or falling.

Next enter the voltage threshold at which you consider the edge occurring. This value can be anywhere between -10 and 10 and can include decimal places.

Finally, choose how many edges you need to find consecutively. If you want to find areas that have NO edges you can specify 0.

Below shows a search that finds all rising edges of the analog signal CH1 with a threshold set at 2.0 Volts. Since the waveform display below is zoomed out the match is shown as simply a line. Zooming in on the search will show the details.



## BUS DATA

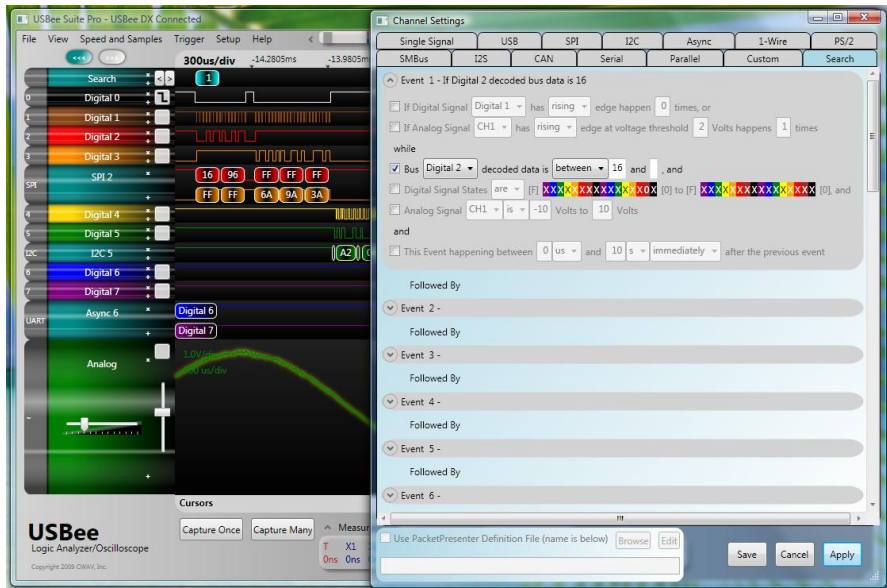
The next line lets you find values of decoded bus data if your waveform display includes them. You can specify specific values to find, values to exclude, or ranges of values.

First select the decoded bus channel you want to search using the first dropdown box.

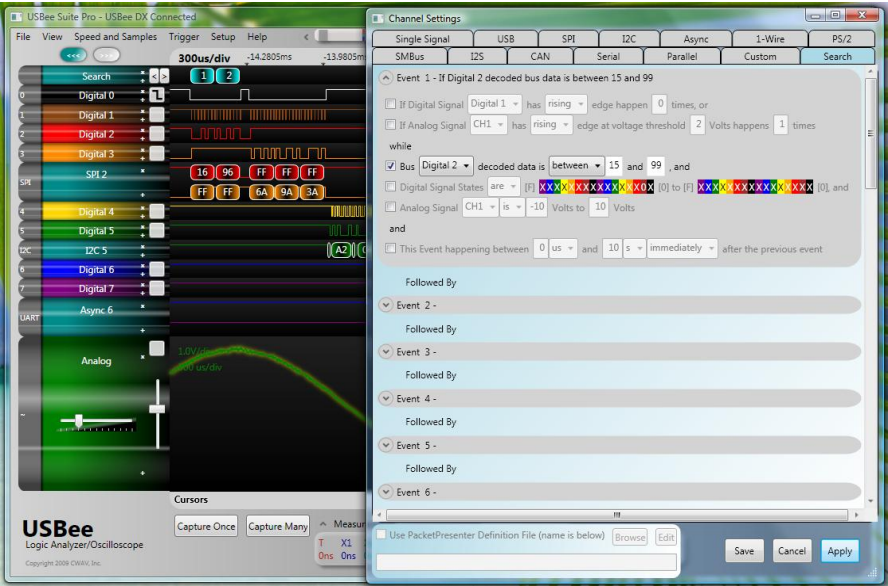
Then choose if you want to find values between (equals) or not between (not equals) the value(s) that follow.

Next enter the actual value of decoded data you are interested in. If it is a single value, leave the second field blank. If you are interested in a range of values, specify the end of the range in the second field.

Below shows a search that finds all decoded data that equals a fixed value of 16.



Below shows a search that finds all decoded data that is between 15 and 99.



## DIGITAL SIGNAL STATES AND RANGES

The next line lets you find states, or ranges of states, on the digital signals.

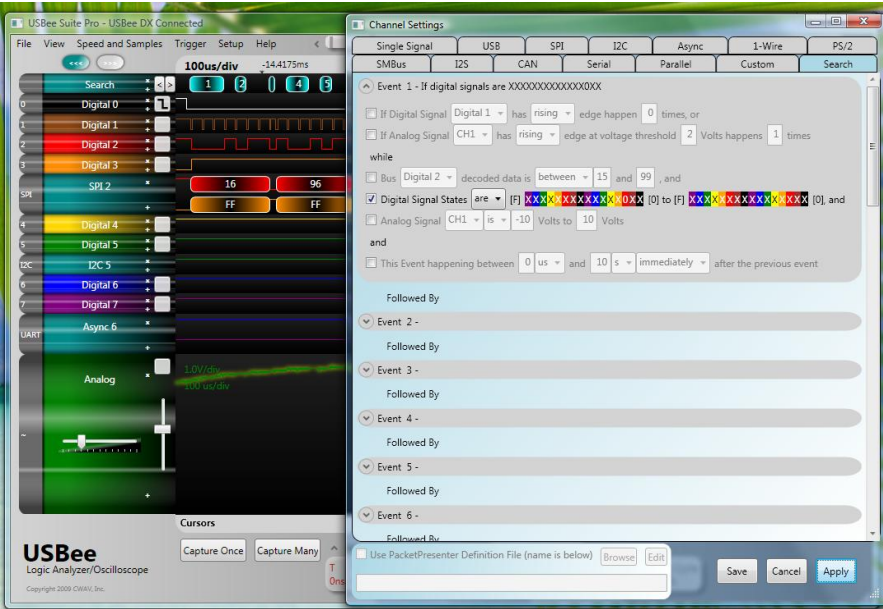
First select if you are looking for the digital states to include the values (**are**), or not include the values (**are not**), that follow.

Then specify the states of all of the digital lines by clicking on each individual signal to change from 0, 1, and X (don't care).

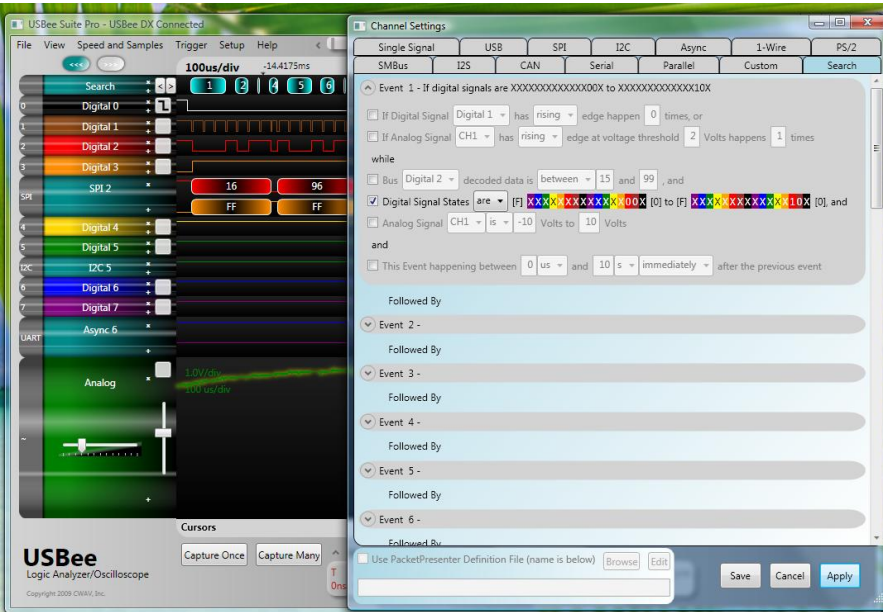
If you are looking for a single set of digital states, leave the second set of signals at all X.

If you want to find a range of states, enter the ending value as the second set of signals. The matched range will then include all values from the first set to the second set, inclusive.

Below shows a search that finds all occurrences of when the Digital 2 signal is logic 0.



Below shows a search that finds all occurrences of when the Digital 2 and Digital 3 signals are 0-0, 0-1, and 1-0. The range starts at the first setting and increments until the second setting.



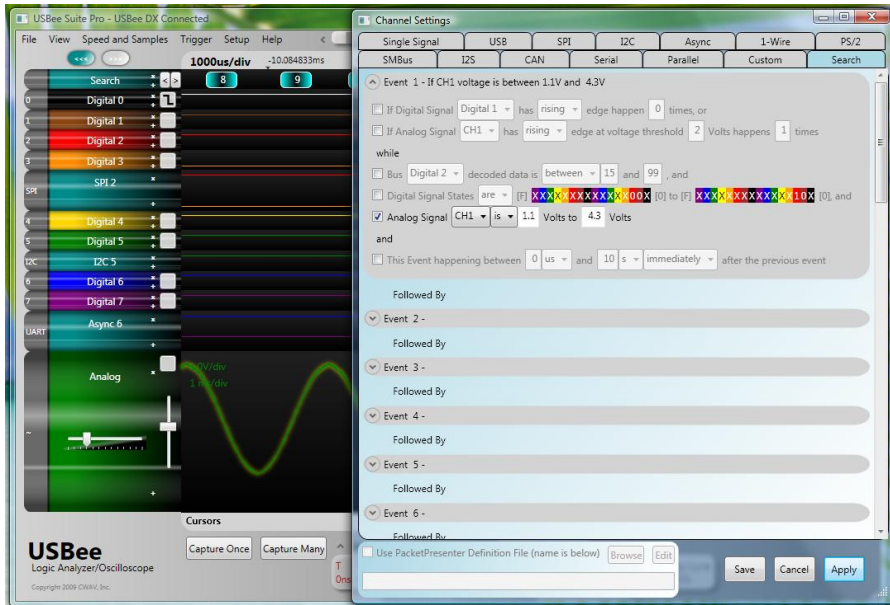
## ANALOG SIGNAL STATES AND RANGES

The next line lets you find voltage ranges on the analog signals, if your USBee has them.

First select if you are looking for the analog voltages to include the values (**is**), or not include the values (**is not**), that follow.

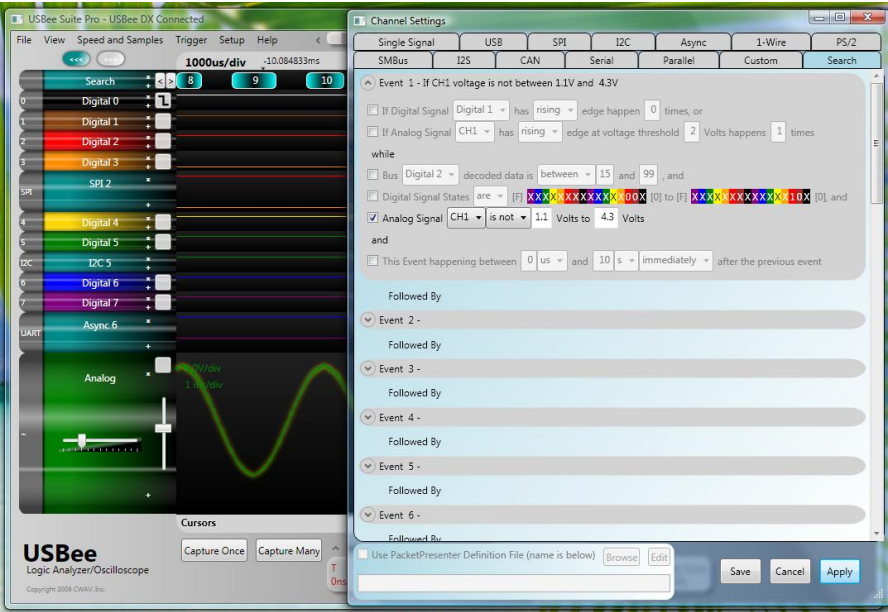
Then specify the voltage range start and end in volts. The matched range will then include all values from the first set to the second set, inclusive.

Below shows a search that finds all times that the analog signal CH1 is between 1.1V and 4.3V.





Below shows a search that finds all times that the analog signal CH1 is NOT between 1.1V and 4.3V.





## TIME WINDOW QUALIFIER

The final line of each Event specifies a time window for the event to occur.

There are two selections to specify a time window: **immediately** or **anytime**.

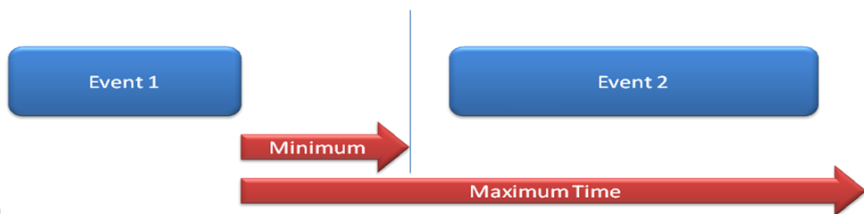
For the first Event, only **anytime** is available. This specifies a time period in which the entire event must occur for a match to be found. For example, if you want to find when two falling edges occur within 10usecs of each other, specify: *This Event happening between 0s and 10us anytime*.

For the Events 2-32, both **anytime** and **immediately** are available.

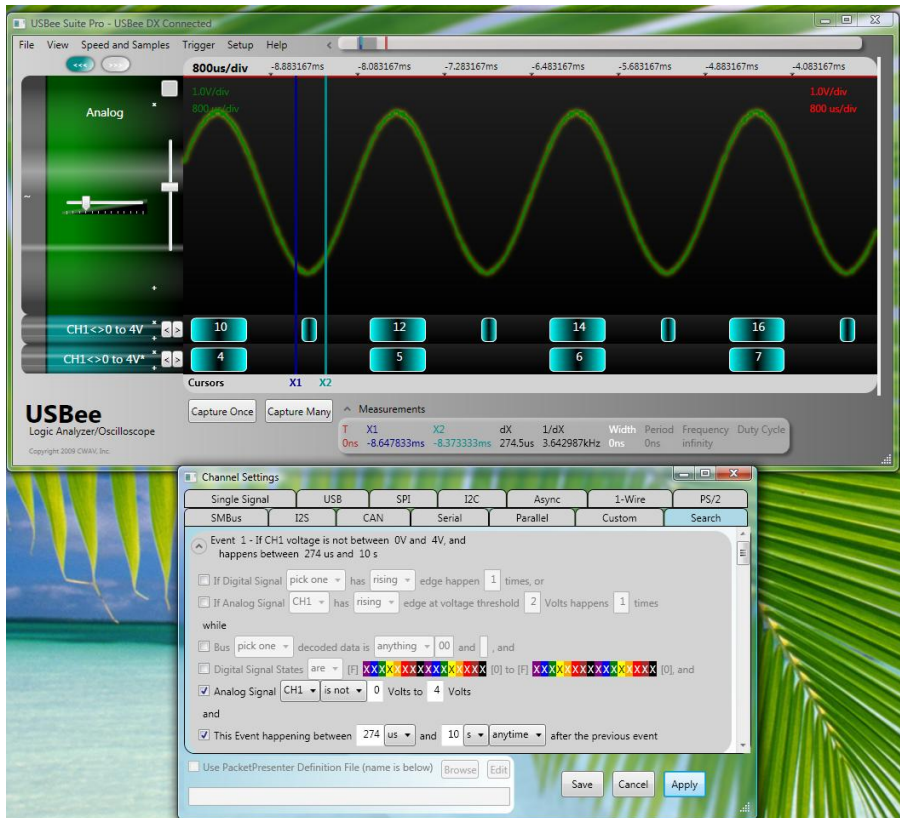
**Anytime** specifies a time period minimum and maximum in which the entire event must occur for a match to be found. The match can occur anytime after the previous event, but must occur within the time window. For example, if you want to find when two falling edges occur within 10usecs of each other, specify: *This Event happening between 0s and 10us anytime after the previous event*. Below shows a diagram for the **anytime** setting. As long as Event 2 is at least Minimum and at most Maximum time it is considered a match.



**Immediately** specifies a time period relative to the end of the previous event in which the entire event must occur for a match to be found. For example, if you want to find when two falling edges occur within 10usecs of each other immediately following the previous event, specify: *This Event happening between 0s and 10us immediately after the previous event*. Below shows a diagram for the **immediately** setting. As long as Event 2 occurs between the Minimum time and the Maximum time it is considered a match.

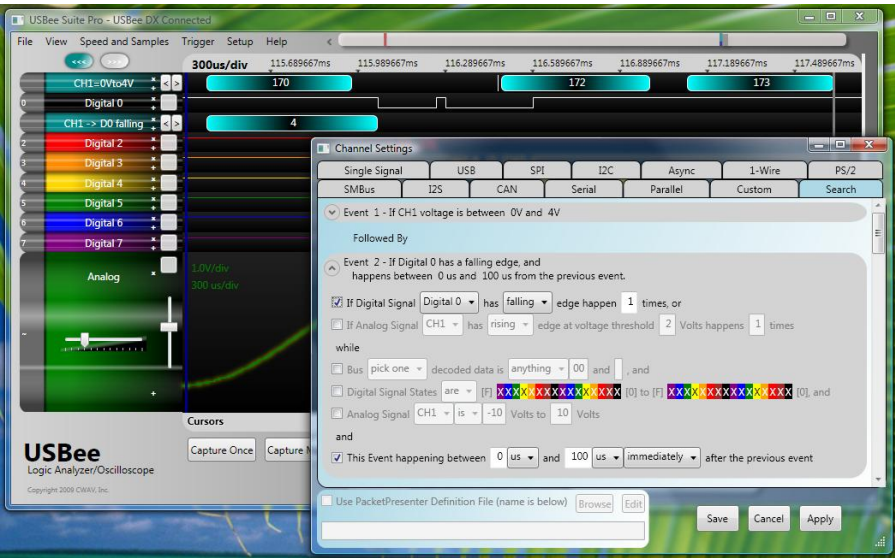


Below shows an example of applying a time window. The first search line shows all occurrences of when the analog CH1 is outside the range 0V to 4V. The second search line is the same criteria with the addition of a time window applied. Only matches that are between 274us and 10seconds are displayed. If you are looking for occurrences that are less than a specific time set the first entry to 0s and the second to the upper limit.



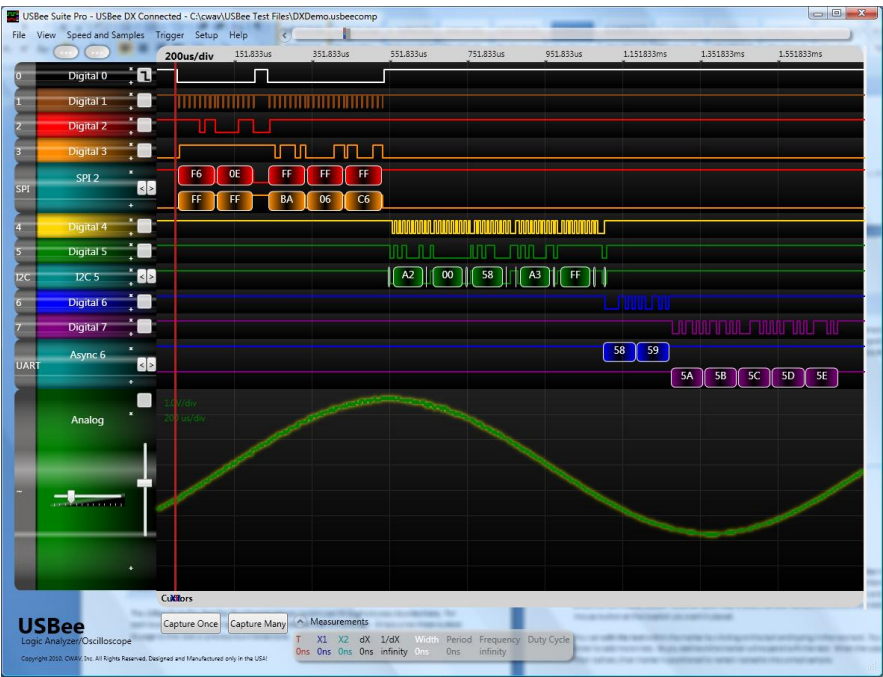
For Events other than the first Event (2-32) you can also specify immediately to indicate that the event must happen between the first time and second time starting at the end of the previous event.

The following example shows two search lines. The first search line shows all occurrences of the CH1 being between 0V and 4V. The second search line shows all occurrences of when the Digital 0 line has a falling edge within 100usecs of the end of the first search.



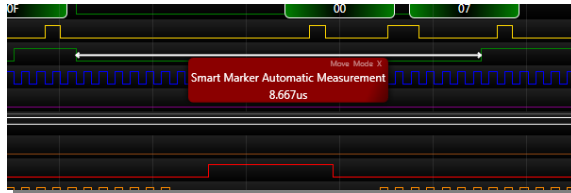
# FAST PAN BUS VIEWING

The USBee RX Suite Fast Pan Bus Viewing lets you quickly pan through a busses decoded data. For each bus there is a left and right pan button on the left side of the screen. Simply press these buttons to page to the next or previous bus transactions.

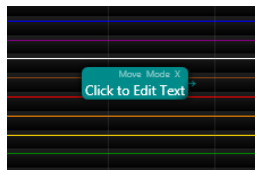


## SAMPLE AND SMART MARKERS

Placing markers in your traces can help detail what is happening in your design. There are two types of markers that can be used. The first marker type locks itself to a sample on a waveform and lets you specify the text. The second is a Smart Marker that automatically measures the pulse width, frequency, period or duty cycle of the waveform at the marker location.



### SAMPLE MARKERS



Markers that are locked to a specific sample are placed using the **View | Add Sample Marker** menu item. Once you select this menu item your cursor changes to 4-way arrows and a Sample Marker moves wherever you move the cursor. To **place the marker**, position it where you want it and then press the left mouse button. Another faster way to place Sample Markers is to press the middle mouse button at the location you want it placed.

You can **edit the text** within the marker by clicking on the text and typing in the new text. You can hit Enter to add more lines. As you add text the marker will expand to fit the text. When the waveforms then redraw, then marker is positioned to remain locked to the correct sample.

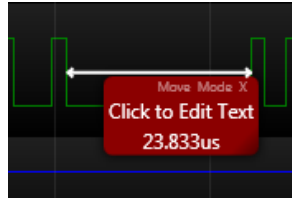
To **move the marker**, click on the Move at the top of the marker. Once you have it moved to the new location use the left mouse button to place it.

The Mode at the top of the marker **changes the direction** of the marker. Click on Mode to change from Left pointing marker to Right pointing marker and vice versa.

To hide all of the markers without deleting them, uncheck the menu item **View | Show Marker Labels**. To turn on the markers, make sure this menu item is checked.

To **delete the marker**, click on the X at the top of the marker. You can delete all markers using the menu item **View | Delete All Markers**. This will delete all Sample Markers and Smart Markers.

## SMART MARKERS



Smart Markers are locked to a specific sample and measure the waveform underneath. They are placed using the **View | Add Smart Marker** menu item. Once you select this menu item your cursor changes to 4-way arrows and a Smart Marker moves wherever you move the cursor. To place the marker, position it where you want it and then press the left mouse button.

You can **edit the text** within the marker by clicking on the text and typing in the new text. You can hit Enter to add more lines. As you add text the marker will expand to fit the text. When the waveforms then redraw, then marker is positioned to remain locked to the correct sample.

To **move the marker**, click on the Move at the top of the marker. Once you have it moved to the new location use the left mouse button to place it.

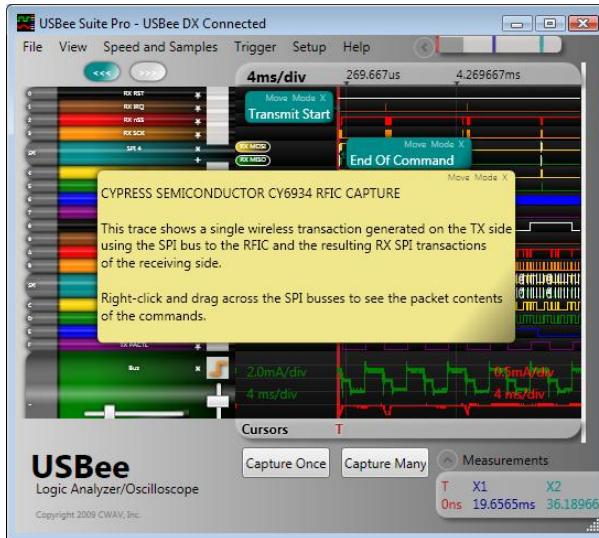
The Mode at the top of the marker **changes the measure mode** of the marker. Click on Mode to cycle through No Measurement, Width, Frequency, Period, and Duty Cycle. An arrow shows the measured area and the measurement shows up as the last line of the marker.

To hide all of the markers without deleting them, uncheck the menu item **View | Show Marker Labels**. To turn on the markers, make sure this menu item is checked.

To **delete the marker**, click on the X at the top of the marker. You can delete all markers using the menu item **View | Delete All Markers**. This will delete all Sample Markers and Smart Markers.

## ANNOTATIONS AND STICKY NOTES

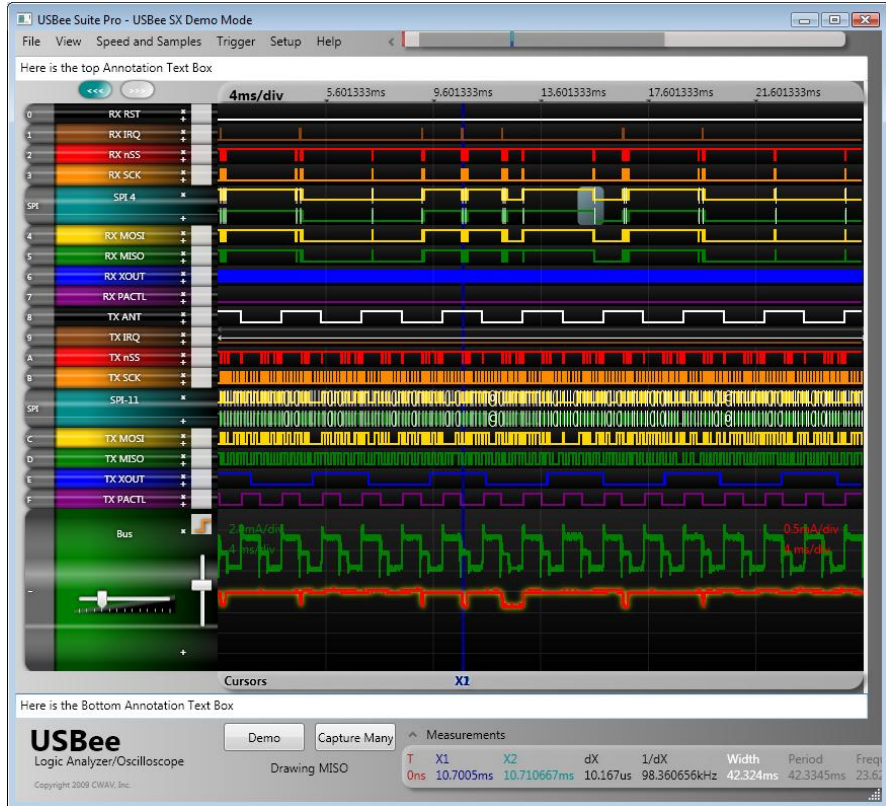
The USBee RX Suite adds Sticky Notes which you can use to further detail your traces for documentation purposes. You can also add Title and Footer text to your display that is saved with the trace file.



## ANNOTATIONS

Annotation Text Boxes are editable text blocks that are located at the Top and Bottom of the USBee RX Suite window. Annotation Text Boxes are enabled and disabled using the **View | Annotation Text Boxes** menu item.

To edit the text, simply select the box and edit the text. This text is then saved with your capture files.





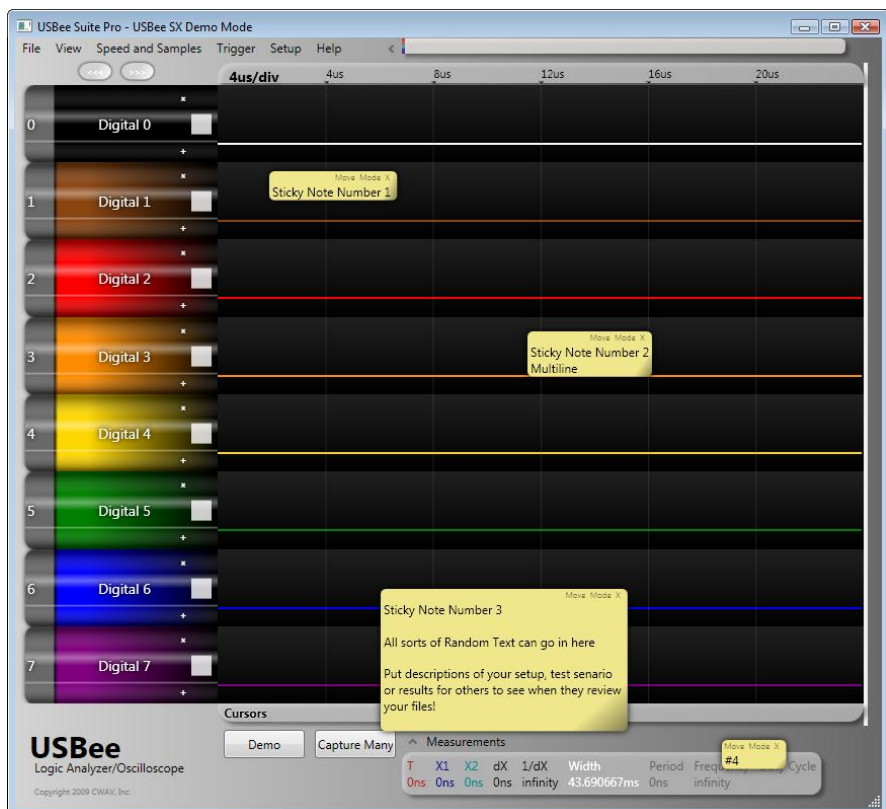
## STICKY NOTES

Sticky Notes are editable text blocks that look like sticky notes. They can be positioned anywhere in the application window. Sticky Notes are placed using the **View | Add Sticky Note** menu item. Once you select this menu item your cursor changes to 4-way arrows and a Sticky Note moves wherever you move the cursor. To place the note, position it where you want it and then press the left mouse button.

You can **edit the text** within the Sticky Note by clicking on the text and typing in the new text. You can hit Enter to add more lines. As you add text the marker will expand to fit the text.

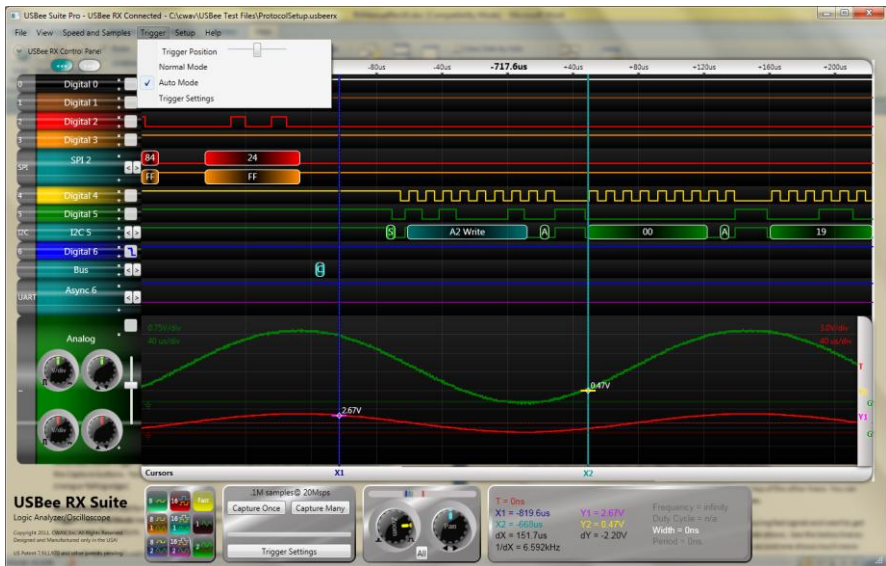
To **move the Sticky Note**, click on the Move at the top of the note. Once you have it moved to the new location use the left mouse button to place it.

To **delete the Sticky Note**, click on the X at the top of the note. You can delete all Sticky Notes using the menu item **View | Delete All Sticky Notes**.



## ACQUISITION CONTROL

The USBee RX Suite adds more trace acquisition and triggering controls such as Normal Mode, Automatic Mode, Single Capture and Multiple Capture.



When the USBee RX Suite is first started, no acquisition is taking place. You need to press one of the acquisition buttons, **Capture Once** or **Capture Many**, at the bottom of the window to capture data.

The **Capture Many** button performs an infinite series of traces, one after the other. This lets you see frequent updates of what the actual signals are doing in real time. If you would like to stop the updating, just press the same button again (now reading Stop) and the updating will stop. This mode is great for signals that repeat over time.

The **Capture Once** button captures a single trace and stops. This mode is good for detailed analysis of a single event, rather than one that occurs repeatedly.

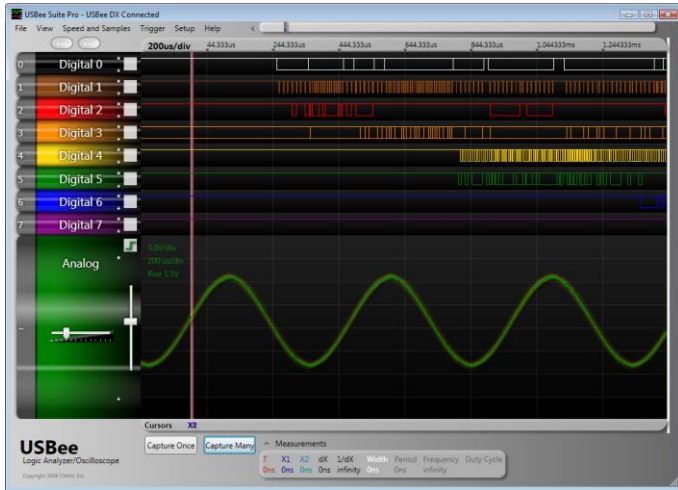
The USBee RX Suite adds the ability to have either Normal Mode (default in the Standard version) or Automatic Mode triggering. This determines when the signals start getting sampled once you press the Capture buttons. You specify the trigger event by selecting one of the USBee signals to trigger on (rising or falling edge).

**Normal Mode** will wait for the trigger event to occur before capturing. Select this option using the **Trigger | Normal Mode** menu item. If the trigger event does not occur you can press the Stop button to terminate the capture.

**Automatic Mode** will wait a set time for the trigger and will automatically trigger if it is not found. Select this option using the **Trigger | Automatic Mode** menu item. If the trigger event does not occur within a specified time, it will automatically start a capture of whatever is on the signals at the time. You can press the Stop button to terminate the capture.

## DISPLAY MODES

The USBee RX Suite lets you widen the trace waveforms, display the analog waveforms as vectors or single sample points, and persist the display from one trace to the next.

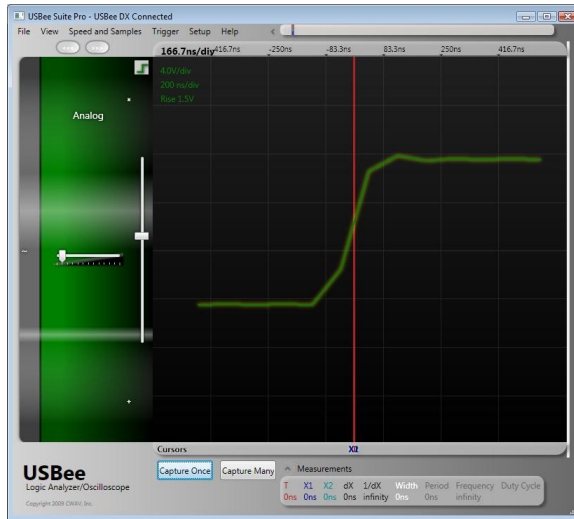


The **Wide** setting shows the waves using a wider pixel setting. This makes the waves easier to see. You can toggle this setting using the menu item **View | Wide Lines**.

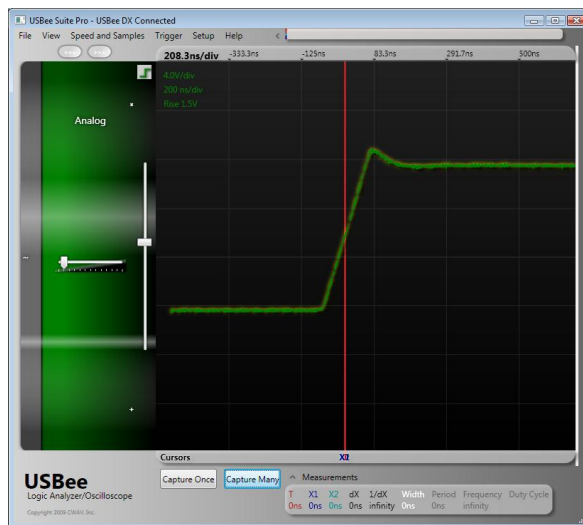
The **Vectors** setting draws the waveforms as a line between adjacent samples. With this mode turned off, the samples are shown simply as dots on the display at the sample position. You can toggle this setting using the menu item **View | Vector-based Waveforms**.

The **Persist** mode does not clear the display and writes one trace on top of the other trace. You can toggle this setting using the menu item **View | Waveform Persistence**.

The benefits of these display modes can be seen when you are measuring fast signals and want to get more resolution out of the oscilloscope than the maximum sample rate allows. See the below traces to see the difference. Each trace is taken of the same signal, but the second one shows much more wave detail over a short time of display updates.



Persist = OFF, Vectors = ON, Wide = ON



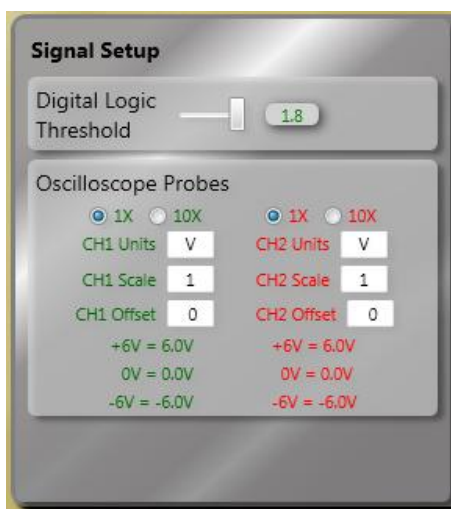
Persist = ON, Vectors = OFF, Wide = ON

## ANALOG CHANNELS SCALING

The USBee RX Suite provides a scaling ability to convert the analog voltages into other units of measurement.

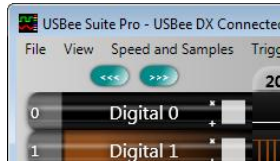
By default, each analog channel is set to display the measurements in Volts where 1V is shown as 1V on the display. Sometimes the measurement might actually mean a different thing than voltage. The menu item **Setup | Analog Channel Settings** lets you specify the units of measurement as well as a scale factor.

Below shows the default setting for the analog channels showing a gain value of 1, offset of 0 and units of Volts.



## BROWSER-LIKE NAVIGATION

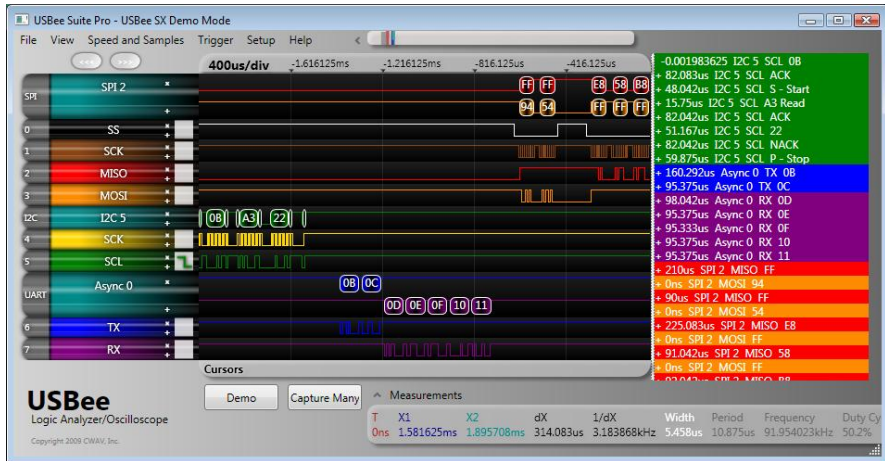
The USBee RX Suite adds browser-like Forward and Back buttons that let you quickly navigate through your trace display.



Each time you stop at a certain point when viewing your waveforms, the location is saved to the history buffer. This allows you to quickly jump back to the previous locations within your trace without having to scroll, pan or zoom. Press the Back Button (Cyan oval with <<<) to go backwards in the history buffer. Press the Forward Button (Cyan oval with >>>) to go forward in the history buffer.

## RELATIVE TIME DECODE

The USBee RX Suite also adds a Relative Time or Absolute Time setting for the decoded data lists.



**Absolute Timestamps** display the sample time for each decoded bus transaction relative to the trigger location. You can turn on Absolute Timestamps using the menu item **View | Decoder Timestamps Absolute**.

```
0.017596417,SPI 2,MISO,4A
0.017596417,SPI 2,MOSI,FF
0.017688458,SPI 2,MISO,AA
0.017688458,SPI 2,MOSI,FF
0.017802750,I2C 5,SCL,S - Start
0.017818458,I2C 5,SCL,A2 Write
0.017900542,I2C 5,SCL,ACK
0.017933167,I2C 5,SCL,00
0.018015250,I2C 5,SCL,ACK
0.018049208,I2C 5,SCL,4A
0.018131292,I2C 5,SCL,ACK
0.018179375,I2C 5,SCL,S - Start
0.018195083,I2C 5,SCL,A3 Read
0.018277167,I2C 5,SCL,ACK
0.018328333,I2C 5,SCL,61
0.018410417,I2C 5,SCL,NACK
0.018470208,I2C 5,SCL,P - Stop
0.018633208,Async 0,TX,4A
0.018728542,Async 0,TX,4B
0.018826583,Async 0,RX,4C
0.018921958,Async 0,RX,4D
0.019017333,Async 0,RX,4E
0.019112708,Async 0,RX,4F
0.019208042,Async 0,RX,50
0.019417042,SPI 2,MISO,FF
0.019417042,SPI 2,MOSI,16
0.019509083,SPI 2,MISO,FF
0.019509083,SPI 2,MOSI,96
0.019733167,SPI 2,MISO,6A
```

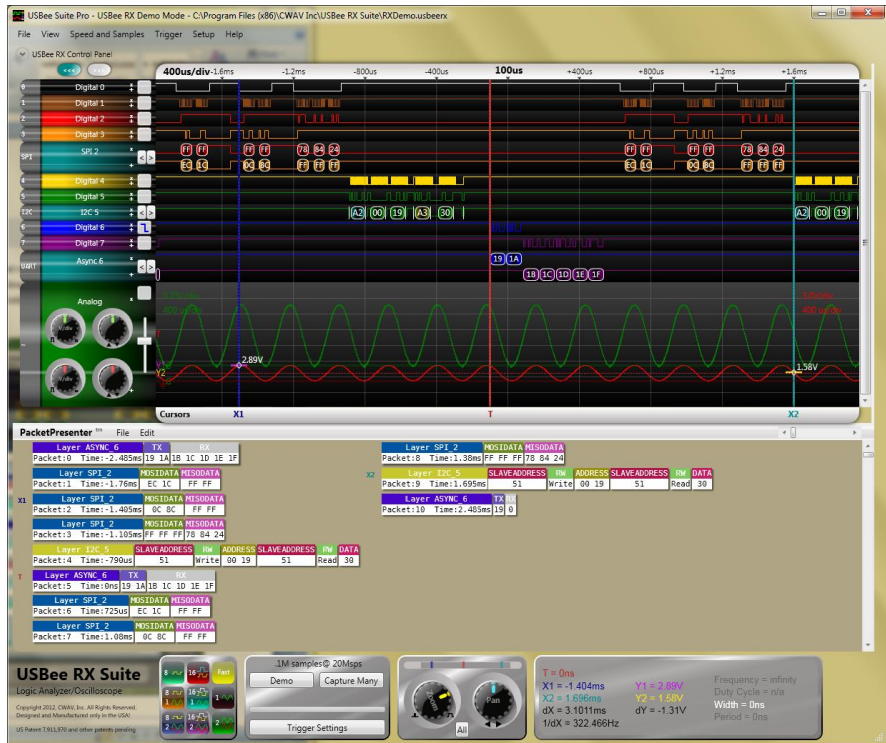
**Relative Timestamps** display the time difference since the last bus transaction and the current transaction. You can turn on Relative Timestamps using the menu item **View | Decoder Timestamps Relative**.

```
+82.042us,I2C 5,SCL,ACK
+51.167us,I2C 5,SCL,22
+82.042us,I2C 5,SCL,NACK
+59.875us,I2C 5,SCL,P - Stop
+160.292us,Async 0,TX,0B
+95.375us,Async 0,TX,0C
+98.042us,Async 0,RX,0D
+95.375us,Async 0,RX,0E
+95.333us,Async 0,RX,0F
+95.375us,Async 0,RX,10
+95.375us,Async 0,RX,11
+210us,SPI 2,MISO,FF
+0ns,SPI 2,MOSI,94
+90us,SPI 2,MISO,FF
+0ns,SPI 2,MOSI,54
+225.083us,SPI 2,MISO,E8
+0ns,SPI 2,MOSI,FF
+91.042us,SPI 2,MISO,58
+0ns,SPI 2,MOSI,FF
+92.042us,SPI 2,MISO,B8
+0ns,SPI 2,MOSI,FF
+114.292us,I2C 5,SCL,S - Start
+15.708us,I2C 5,SCL,A2 Write
+82.083us,I2C 5,SCL,ACK
+32.667us,I2C 5,SCL,00
+82.042us,I2C 5,SCL,ACK
+33.958us,I2C 5,SCL,12
+82.042us,I2C 5,SCL,ACK
+48.083us,I2C 5,SCL,S - Start
+15.708us,I2C 5,SCL,A3 Read
+82.083us,I2C 5,SCL,ACK
+51.167us,I2C 5,SCL,29
```



## PACKETPRESENTER™

The USBee RX Suite adds the PacketPresenter™ feature that runs alongside of the existing bus decoders. The PacketPresenter™ takes the output of raw binary data from the bus decoders and parses the stream according to users PacketPresenter Definition File for the intent of displaying the communications in easily understood graphical displays.



## OVERVIEW

Using the USBee RX Suite application, it is normal for users to debug communication that is being transmitted between ICs or system components. This debugging can be performed by viewing the waveforms on the screen, or by viewing decoded bus traffic for the various types of busses. For example users can see the voltage versus time waveforms of an ASYN bus Tx and Rx lines, or decode the waveform into a byte stream using the standard bus definition (ASYN for example) that is then displayed in text in-line with the waveform.

The PacketPresenter™ feature runs alongside of the existing bus decoders of the USBee RX Suite. The PacketPresenter™ takes the output of raw binary data from the bus decoder and parses the stream according to users PacketPresenter Definition File for the intent of displaying the communications in

easily understood graphical displays. The resulting packets are then displayed in a window below the waveforms.

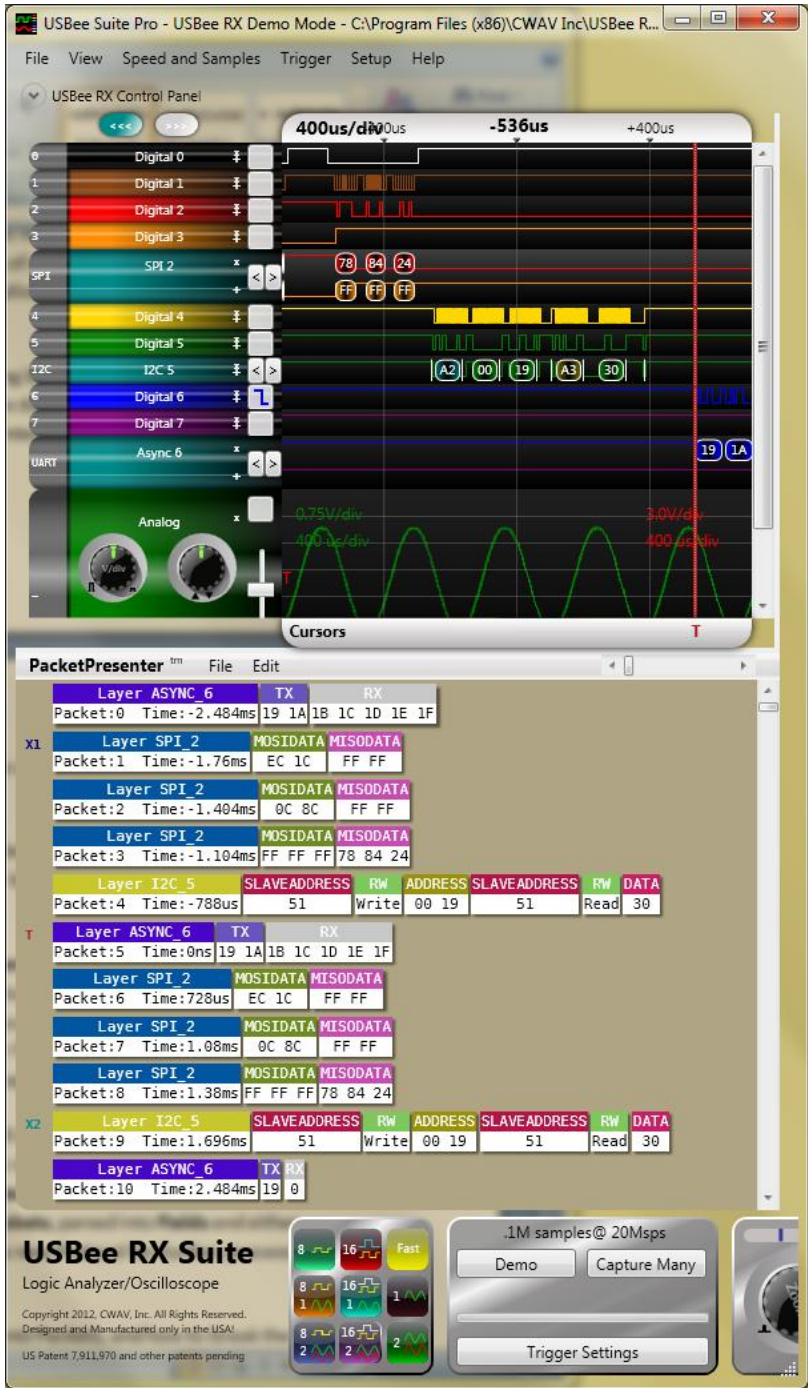
To **Show the PacketPresenter** window, choose the menu item View | Show PacketPresenter Window. To resize the window, click and drag the thin bar between the PacketPresenter window and the waveforms.

Protocols are defined using a text file, called a **PacketPresenter Definition File**, which specifies the fields within the protocol and how to display that information on the screen. It is intended to be generic enough that customers can create their own protocol decoders for their own custom bus types. Initially, all bus types use a generic PacketPresenter file to display the bus data for that bus. You can then create and customize your own PacketPresenter file for any bus.

It is assumed that each **PacketPresenter Definition File** will correspond to one single bus type, and that the incoming bytes from that bus will be inputs for the decoding process. This stream of data is called an incoming **Data Stream** and it is handled by a **Protocol Processor**. Each Protocol Processor takes a single incoming Data Stream that is broken into **Packets**, parsed into **Fields** and either displayed as a field on the screen, ignored, and/or sent to a new Protocol for further processing (as in an N layer protocol).

Each Protocol Processor defines how to break the stream into Packets, and how to break the Packets into Fields. These Fields can then be displayed or sent to another Data Stream for further processing.

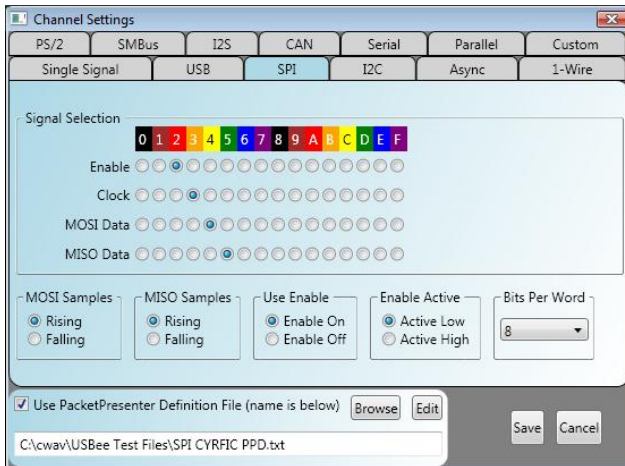
Below shows a sample PacketPresenter output screen.



---

## SETTING UP THE PACKETPRESENTER

Each digital waveform on the screen can be defined as a different bus (I2C, SPI, etc.) in the Channel settings dialog box by clicking on the white box to the left of the signal name. Below shows the Channel Settings dialog box.



All busses will use a default PacketPresenter internal file to display the bus data until a custom file is defined by the user.

To enable a custom PacketPresenter file for this channel, check the **“Use PacketPresenter definition file (name is below)”** checkbox. Then choose the PacketPresenter definition file by clicking the **Browse** button to the right. Once you choose the file, you can edit the contents by clicking the **“Edit File”** button.

Once the PacketPresenter is enabled all bus decodes will be processed through the PacketPresenter as well as the original bus decoder.

---

## VIEWING THE PACKETPRESENTER OUTPUT

The PacketPresenter output is shown in a window below the waveform screen. This window can be shown using the menu item **View | Show PacketPresenter Window**, or by clicking and dragging the thin bar between the waveform and PacketPresenter window.

## SAVING PACKETPRESENTER DATA TO TEXT OR RTF FILES

The PacketPresenter output can be saved to either a Text file or an RTF file (Rich Text Format). The text file output is a textual representation of the packets as seen below. Access these features through the **File | Save As Text** or **File | Save As RTF** menu items.

```
Layer: CYPRESSRFIC    DIR      INC      ADDRESS      READDATA
Time: 615.2797ms      Read    False   CHANNEL_ADR  0

Layer: USBBUS         PID      ADDR    EP      PID      INDATA      HS
Time: 616.0198ms      IN       2       0       DATA0   22 2A 00 07 05 81 03 08  ACK

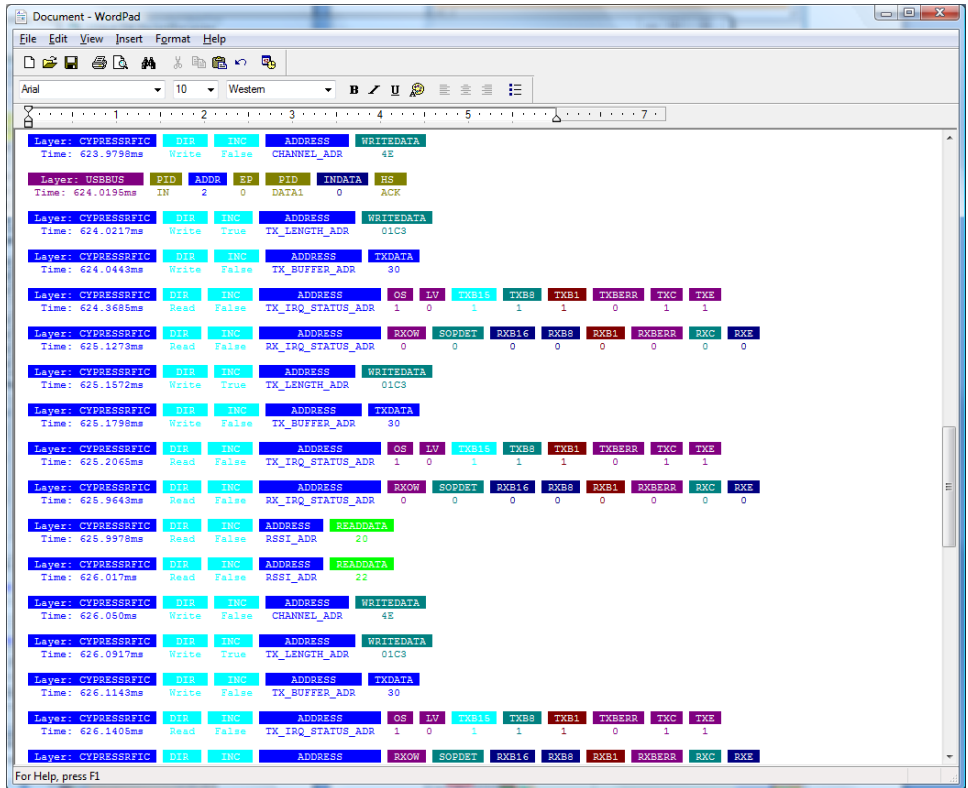
Layer: USBBUS         PID      ADDR    EP      PID      INDATA      HS
Time: 617.0197ms      IN       2       0       DATA1   00 0A 09 04 01 00 01 03  ACK

Layer: USBBUS         PID      ADDR    EP      PID      INDATA      HS
Time: 618.0197ms      IN       2       0       DATA0   01 02 00 09 21 11 01 00  ACK

Layer: USBBUS         PID      ADDR    EP      PID      INDATA      HS
Time: 619.0197ms      IN       2       0       DATA1   01 22 D1 00 07 05 82 03  ACK

Layer: USBBUS         PID      ADDR    EP      PID      INDATA      HS
Time: 620.0197ms      IN       2       0       DATA0   0A0008  ACK
```

Saving data to an RTF file format saves the graphical nature of the packets and can be read by many word processing programs, such as Microsoft Word and WordPad. Below is a screenshot of data saved to an RFT file and viewed using WordPad.



In order to maintain correct position of the graphical portions of the RTF file, all spaces are converted to the character "~" and set to the background color. Viewed or printed in the RTF format will look correct as above. If you copy only the text of this output, you will want to search and replace every "~" with a space.

## COPYING PACKETPRESENTER OUTPUT TO OTHER PROGRAMS

You can copy the contents of the PacketPresenter output window to other programs in a number of ways.

First, you can copy the screenshot of the window by selecting the window and pressing Alt-PrtScr on your keyboard. This copies the image of the window to the Windows clipboard and you can paste that image into any program that accepts images.

You can also use the **Edit | Copy as Text** or **Edit | Copy as RTF** menu items. All packets are copied to the clipboard in the format specified. Below is a sample of packets copied in RTF format and pasted into Word.

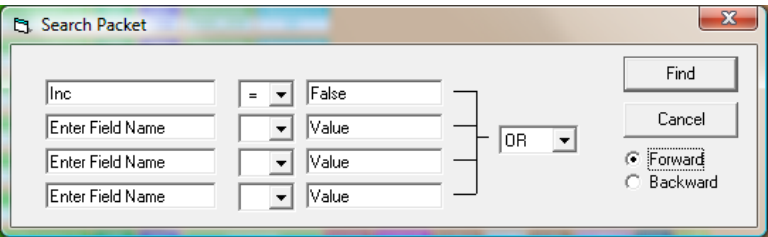
Layer: CYPRESSRFC	DIR	INC	ADDRESS	WRITEDATA								
Packet: 1 Time: 7.339333ms	Write	False	CHANNEL_ADR	0								
Layer: CYPRESSRFC	DIR	INC	ADDRESS	WRITEDATA								
Packet: 2 Time: 7.347833ms	Write	False	RX_CTRL_ADR	82								
Layer: CYPRESSRFC	DIR	INC	ADDRESS	READDATA								
Packet: 3 Time: 7.356833ms	Read	False	RSI_ADR	20								
Layer: CYPRESSRFC	DIR	INC	ADDRESS	RXOW	SOPDET	RXB16	RXB8	RXB1	RXBERR	RXC	RXE	
Packet: 4 Time: 9.189167ms	Read	False	RX_IRQ_STATUS_ADR	0	1	0	1	1	0	1	1	
Layer: CYPRESSRFC	DIR	INC	ADDRESS	RXDATA								
Packet: 5 Time: 9.198833ms	Read	False	RX_BUFFER_ADR	08 82 1E 99 A7 28 2A 8A 12 88 9E 58 18 CA C0 C0								
Layer: RXDATA	RECEIVEDATA											
Packet: 6 Time: 9.204667ms	08 82 1E 99 A7 28 2A 8A 12 88 9E 58 18 CA C0 C0											
Layer: CYPRESSRFC	DIR	INC	ADDRESS	READDATA								
Packet: 7 Time: 9.265167ms	Read	False	CHANNEL_ADR	0								

Below is a sample of packets copied in Text format and pasted into Notepad.

Layer: CYPRESSRFC	DIR	INC	ADDRESS	WRITEDATA												
Packet: 1 Time: 7.339333ms	Write	False	CHANNEL_ADR	0												
Layer: CYPRESSRFC	DIR	INC	ADDRESS	WRITEDATA												
Packet: 2 Time: 7.347833ms	Write	False	RX_CTRL_ADR	82												
Layer: CYPRESSRFC	DIR	INC	ADDRESS	READDATA												
Packet: 3 Time: 7.356833ms	Read	False	RSI_ADR	20												
Layer: CYPRESSRFC	DIR	INC	ADDRESS	RXOW	SOPDET	RXB16	RXB8	RXB1	RXBERR	RXC	RXE					
Packet: 4 Time: 9.189167ms	Read	False	RX_IRQ_STATUS_ADR	0	1	0	1	1	0	1	1					
Layer: CYPRESSRFC	DIR	INC	ADDRESS	RXDATA												
Packet: 5 Time: 9.198833ms	Read	False	RX_BUFFER_ADR	08 82 1E 99 A7 28 2A 8A 12 88 9E 58 18 CA C0 C0												
Layer: RXDATA	RECEIVEDATA															
Packet: 6 Time: 9.204667ms	08 82 1E 99 A7 28 2A 8A 12 88 9E 58 18 CA C0 C0															
Layer: CYPRESSRFC	DIR	INC	ADDRESS	READDATA												
Packet: 7 Time: 9.265167ms	Read	False	CHANNEL_ADR	0												

## SEARCHING FOR PACKETS

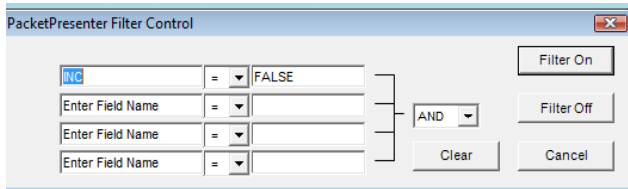
Once displayed, you can search for the next packet that contains certain fields that match your criteria. Below is the Search Packet dialog box that is shown by using the **View | Packet Search** menu item.



In the leftmost textboxes, type the Field Label. Then select the comparator operator (equals, not equals, less than, greater than...) and finally the value that the field is to be compared against. Finally, if there is more than one field in the search list, choose whether to AND or OR the search terms. When you click Find, the next packet in the list (starting from the top of the window) will be placed at the top of the window. You can search forward or backward by selecting the appropriate radio button on the right.

## FILTERING PACKETS

Once displayed, you can filter the output to only show packets that contains certain fields that match your criteria. Below is the Filter Packet dialog box that is shown by selecting the **View | Packet Filter** along with the resulting PacketPresenter output.



In the leftmost textboxes, type the Field Label. Then select the comparator operator (equals, not equals, less than, greater than...) and finally the value that the field is to be compared against. Finally, if there is more than one field in the search list, choose whether to AND or OR the search terms. When you click **Filter On**, only the packets matching the criteria are displayed. To turn off the filtering, click on the **Filter Off** button.

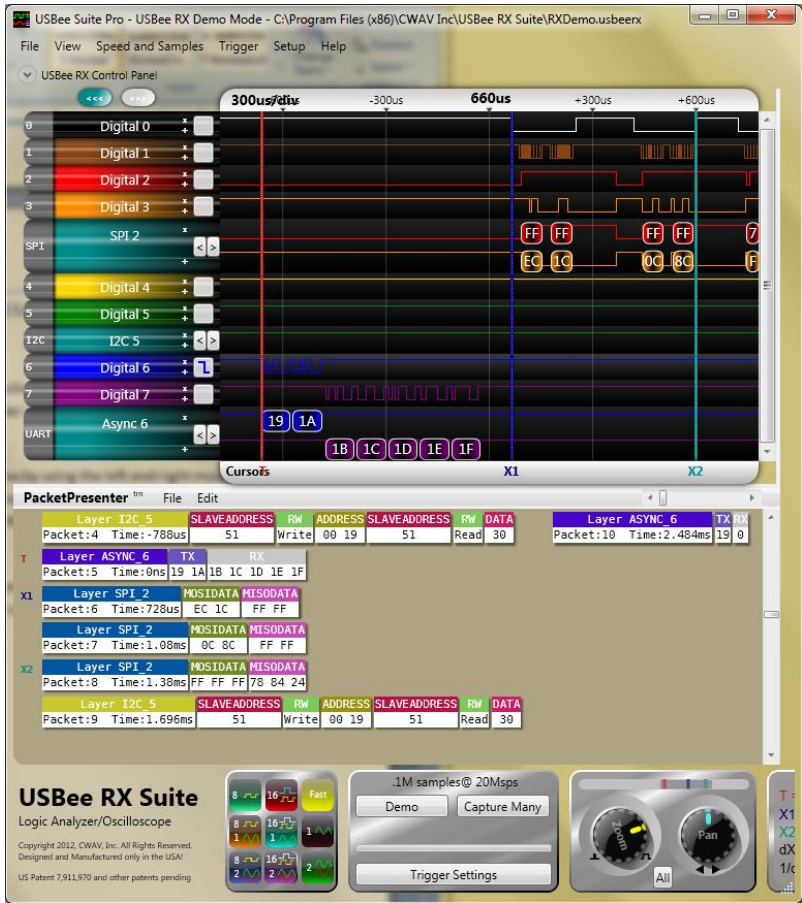


# PACKETPRESENTER TO WAVEFORM ASSOCIATION USING CURSORS

The PacketPresenter output and waveform display are synchronized using the Cursors. This feature allows you to correlate what is shown in the PacketPresenter window to the actual waveform on the logic analyzer that created that packet.

You can place the cursors using the PacketPresenter window by using the left and right mouse buttons. Place the mouse over the packet you want to place the cursor on and click the left or right mouse button. The cursors are placed at the beginning of the packets. The resulting difference between cursors is shown in the Measurement Window.

Conversely, if you place the X1 and X2 cursor in the Waveform display using the Cursors Bar, the cursors are moved in the PacketPresenter window and it is scrolled to show that location in the packet list.



---

## PACKETPRESENTER DEFINITION FILE FORMAT

Each PacketPresenter Definition file defines how the incoming data stream is represented in the PacketPresenter screen of the USBee RX Suite application. These PacketPresenter Definition files are in text format and are easily created using either a simple text editor.

Each bus defined in the USBee RX Suite application can have a different PacketPresenter Definition File.

The intent of the PacketPresenter is to produce a series of 2 dimensional arrays of labels and values to be displayed as below by the user interface.

Command	Length	Address	Data
45	2	84DF	34

Command	Value
Read RSSI	14.34

Command	Setting
23	Power Amp On

It is the PacketPresenter Definition File that defines how the data is to be parsed and displayed.

---

### COMMENTS IN THE PACKETPRESENTER DEFINITION FILE

Comments are started with a semicolon (;) and go until the end of the line.

---

### CONSTANTS IN THE PACKETPRESENTER DEFINITION FILE

Constants are fixed numbers anywhere in the file. These constants can be expressed as decimal, hex, or binary using suffixes after the value. Decimal has no suffix. Hex uses the suffix "h". Binary uses the suffix "b".

So,

```
16 = 10h = 10000b  
244 = F4h = 11110100b
```

Gain and offset values used in the Fields section are always in decimal and can contain decimal places.

---

## PACKETPRESENTER DEFINITION FILE SECTIONS

Each PacketPresenter Definition File has the following syntax that separates the file into sections that correspond to the Channel definition and each of the Protocol Processors.

```
[Protocol]
. . .
[Protocol]
. . .
[Protocol]
. . .
```

---

### PROTOCOL SECTION

Each Protocol Section defines what the incoming data stream looks like, how to break the data stream into packets, and how to parse out the fields in each of the packets. Multiple Protocol Sections can be defined for passing data from one Protocol Section to another.

Each Protocol Section has the following syntax that specifies the packetizing and parsing into fields.

```
[Protocol]
name = ProtocolName
[Packet]
    packet processing settings
[Fields]
    packet field processing settings
    packet field processing settings
    packet field processing settings
    . . .
```

The *ProtocolName* is a label that uniquely identifies this protocol processor. This name is used in the Field definitions to define which Protocol to route a field of data (for use by multilayer protocols).

The highest level Protocol is the first protocol in the file. This is the Protocol Processor that is sent the incoming data stream from the bus as defined in the Channel Settings Dialog Box for that waveform.

---

### BYTE-WISE BUSSES VS. BIT-WISE BUSSES

Some busses are by nature byte oriented, while others are bit oriented. The following table shows the type of bus.

#### Bytewise Busses

- Async
- I2C
- Parallel
- SPI
- PS2

#### Bitwise Busses

- Serial
- I2S
- OneWire
- CAN
- USB
- SDIO

## BUS EVENTS

---

Each bus type also can have certain bus events that may be significant in the decoding of a protocol. One such event is an I2C Start Bit. While the Start bit is not an actual bit in the data stream, it does signify to the I2C slave that a certain transaction is taking place. These bus events are inserted into the data stream and can be used (or ignored) by the protocol processors. The list of Bus Events supported is in the following table.

Bus Type	Event
Async	1 – Parity Error
I2C	1 - Start Bit 2 - Stop Bit 4 - ACK 8 – NACK
SPI	1 - SS Active 2 - SS Inactive Note: You MUST have SS On in the channels settings for these events to occur
USB	1 – SETUP/IN/OUT Received 2 –ACK/NACK/Stall Received 4 – No Handshake received
CAN	1 – Start of CAN packet 2 – End Of CAN packet
1-Wire	1 - Reset Found 2 - Presence Found
Parallel	
Serial	
PS/2	1 – Device to Host byte follows 2 – Host to device byte follows
I2S	1 - WordSelect Active 2 - WordSelect InActive
SMBus	1 - Start Bit 2 - Stop Bit

SDIO	1 - Start of Packet
	2 - End of Packet
	3 - APP CMD packet
	4 - R1 response packet
	5 - R2 response packet
	6 - R3 response packet
	7 - R6 response packet
	8 - Data Packet
	9 - R4 response packet
	0 - R5 response packet

A Bus Event of 127 (7Fh) is a special event that occurs at the end of a packet of data that is sent from one protocol to another. This can be used to end the packet sent to the new layer using the [END] section and the type = event in the new protocol level.

## DATA CHANNELS AND MULTIPLE DATA SIGNALS

Some buses can also have more than one data signal used in the protocol. One example of this is the SPI bus, where for each byte sent on the MOSI line there is one byte received on the MISO line. In the protocol definition you can specify which of the signals to expect the next field of data to be sent on. In the SPI example, you may get a Command and Length field on one signal, followed by the read data back on the other signal. The decoder would take that into account and show the command, Length and Data as a single transaction.

Multiple signals are differentiated in the PacketPresenter using the X and Y channel specifiers. These channels are specified by selecting the signals to use for that bus in the Channel Settings dialog box. The following table shows which signals are the X and Y signals.

Bus Type	Channel Setting Dialog Box setup for Channel X	Channel Setting Dialog Box setup for Channel Y	Notes
ASYNC	Least Significant Async Channel selected	Next Least Significant Async Channel selected	If more than 2 Async channels are selected to be decoded, the additional channels are not used by the PacketPresenter.
SPI	Signal chosen for MISO	Signal chosen for MOSI	Data Bytes alternate channels since there is one byte X for every one byte Y
1 Wire	Data Signal	Not used	
I2C	Data on SDA/SCL bus	Not Used	
Parallel	All Data Signals sampled together	Not Used	Each sample of all channels is the data word sent to channel X
Serial	Serial Data	Not Used	
CAN	Rx Data	Not Used	
PS/2	Data from Device to Host	Data from Host To Device	
USB	Data on D+/D- bus	Not Used	The data stream contains the Sync, PIDs, data fields and CRCs.
SDIO	CMD traffic	DAT data traffic	

**Table 1. Channel X and Channel Y Definitions Per Bus Type**

## PACKET SECTION

---

The Packet section defines how a packet is bounded and what, if any, preprocessing needs to be done on the packet before the fields can be processed.

```
[Packet]
[Start]      . . . ; How does a packet start?

[End]        . . . ; How does a packet end?

[Decode]     . . . ; What decoding needs to be
                ; done to get real data?
```

## START AND END SECTIONS

---

The Start and End sections define how a packet is bounded. The available packet bounding Types are defined below:

For [START]

- Next: The next byte or bit is assumed the start of a packet
- Signal: An external signal indicates the start of a packet
- Value: A specific value in the data indicates the start of a packet
- Event: A bus specific bus Event or Events indicates the start of a packet

For [END]

- Next: The next byte or bit is assumed the end of a packet
- Signal: An external signal indicates the end of a packet
- Value: A specific value in the data indicates the end of a packet
- Length: A specific or calculated length determines the end of a packet
- Event: A bus specific bus Event or Events indicates the end of a packet
- Timeout: A packet ends after a set timeout without data or events

## TYPE = NEXT

---

The start or end of a packet is the next byte or bit to arrive.

```
[Packet]
[Start] or [End]
type = Next      ; Start/End of a packet is the
                  ; next byte/bit to arrive
```

## **TYPE = SIGNAL**

---

The start or end of a packet can be indicated by a separate signal (such as a chip select or a frame signal) using the signal setting.

```
[Packet]
[Start] or [End]
type = signal                ; Start/End of a packet is based
                             ; on a signal
signal = signalvalue         ; Signal number 0 - 15
level = 1                    ; level the signal needs to be
```

## **TYPE = VALUE**

---

The start or end of a packet can be indicated by a certain data value contained in the data using the value setting. Multiple values can be used, where any one match starts or ends a packet. All bits in the Value are included in the resulting packet at the start of the packet. You must also specify the number of bits that the value covers (defaults to 8 bits if not specified) using the bits keyword. You can specify a mask value to apply to the start data and values. When the mask value has a bit that is a 1, that bit in the value and data are compared. All values are assumed MSB first.

```
[Packet]
[Start] or [End]
type = value                 ; Start/End of a packet is based on a data value
mask = bitmask               ; Bitmask to apply to the data stream
value = value1                ; value that the data needs to be to start/End
value = value2                ; value that the data needs to be to start/End
value = value3                ; value that the data needs to be to start/End
bits = 8                     ; how many bits in the start/End word
```

You can use the EXCLUDE keyword in the [END] section to leave the end data on the data stream for the next packet. This is useful for when there is no indication of the end of a packet except for the arrival of the next packet.

## **TYPE = LENGTH**

---

Only valid in the [END] section, the end of a packet can be indicated by a certain length of data. You use the BitLength or the ByteLength keywords to specify how long the packet is. The length can either be a fixed length expressed as a constant, or variable length based on the contents of a packet in the data stream.

```
type = length                ; End of a packet is based
                             ; on a length
Bytlength = length           ; How many bytes per
                             ; packet
or
Bitlength = length           ; How many bits per packet
```

To use the contents of one of the fields as the packet length, you use the name of the field defined in the Fields section. You can also do simple arithmetic on the field value to compute the final packet size.



```

type = length      ; End of a packet is based
                  ; on a length
Bytelength = fieldname * 2 + 2
              ; field holding packet size
              ; * (or /) a constant (optional)
              ; + (or -) a constant (optional)

```

If present, the \* or / must come before the + or – offset and is executed first.

For example, if *fieldname* Field has the contents of 16, then the following is true:

$fieldname * 2 + 2 = (16 * 2) + 2 = 34$

$fieldname + 2 = 16 + 2 = 18$

$fieldname / 2 - 2 = (16 / 2) - 2 = 6$

$fieldname / 2 = 16 / 2 = 8$

$fieldname + 2 * 2 = \text{invalid}$  (\* must come before offset)

$fieldname - 2 / 2 = \text{invalid}$  (/ must come before offset)

The length of the packet includes ALL of the data from each of the data channels for that bus. If the bus contains only one data channel (such as I2C), the length counts all data on that one bus. If the bus has two data channels, the length refers to all data on both channels combined.

---

## TYPE = EVENT

The start or end of a packet can be indicated by the reception of any of the bus specific Events. For example in I2C you get a Bus Event for each Start Bit and a Bus Event for each Stop Bit. In USB you get a Bus Event for each Sync word and a Bus Event for each EOP. Available bus types are defined in **Error! Reference source not found..**

The event value is a bitmask that includes all events that you want to use. If any of the events occur, a packet will be started or ended.

```

type = Event      ; Start/End of a packet is
                  ; signaled by event
event = 1          ; Use Event 1. Available events
                  ; depend on bus type
or
event = 3          ; Use either Event 1 or Event 2

```

---

## TYPE = TIMEOUT

The end of a packet is determined by a timeout since the last valid data or event on the bus. The timeout is defined in units of microseconds.

```

[Packet]
[Start]
type = timeout     ; End is after timeout
timeout = 45       ; microseconds since last data/event received

```

## CHANNELX, CHANNELY OR CHANNELXorY

---

CHANNELX, CHANNELY or CHANNELXorY specifies what channel is used when an event or data is defined for starting or ending a packet. Channel X and Channel Y are different based on what the physical bus is and can be found in Table 1. Channel X and Channel Y Definitions Per Bus Type. If it does not matter which channel the data or event occurs on (it could be either), use the CHANNELXorY keyword.

```
[Packet]
[Start]
type = value      ; Start of a packet is based on
                  ; a data value
value = 4lh       ; value of data that starts the
                  ; packet
bits = 8
channelX          ; data/event must be received
                  ; on channel X
or
channelY          ; data/event must be received
                  ; on channel Y
or
channelXorY       ; data/event must be received
                  ; on either channel X or Y
```

## DECODE SECTION

---

Each packet can have encoding on the data that needs to be removed in order to see the real data. This section defines what decoding should be done to the packet. The entire packet from start to end is sent through the decoders. If only select parts of the packet needs to be decoded, you must create your own Add-In decoder using the ADDIN keyword.

Available decoding types are:

Keyword	Definition
NRZI	A bit change on the input means a 1 bit on the output, no change a 0
MANCHESTER	Remove Manchester encoding from data
INVERT	Invert all bits
ZBI5	Zero-Bit Insertion removal (removes the 0 added after 5 1s)
ZBI6	Zero-Bit Insertion removal (removes the 0 added after 6 1s)
ADDIN	Call your own packet decoder using the PacketPresenter API routine APIDecode()
substring	Substitute bytes in the stream (no spaces allowed)

Multiple decoders can be used and are processed in the order listed.

## SUBSTITUTIONS

---

Substitutions allow a sequence of bytes (up to 3) to be replaced with a different set (same size or less) of bytes. They can only be used on bytestreams, not bitstreams. Substrings define the bytes input and the bytes output. The Substrings must not contain any spaces. Examples of this are below:

```
[1]=[2]           ; Replaces all 1s with 2s
[1][2]=[3]        ; Replaces all 1 immediately
                  ; followed by 2 with 3
[1][2]=[3][4]     ; Replaces all 1 immediately
                  ; followed by 2 with 3
                  ; immediately followed by 4
[1][2][3]=[4]     ; Replaces all 1, 2, 3 with 4
[1]=[2][3][4]     ; INVALID, the number of
                  ; output bytes must be less
                  ; than or equal to the input
```

As an example, the HDLC protocol uses the byte value 7Eh as the start and end flag of the packets and replaces all 7Eh in the data with the bytes 7Dh followed by 5Eh. It also replaces all 7Dh in the data with the bytes 7Dh followed by 5Dh. To remove this coding you would use the lines:

```
[7Dh][5Eh]=[7Eh]
[7Dh][5Dh]=[7Dh]
```

## FIELDS SECTION

---

Once the packet is delineated and decoded by the previous sections, it is ready to be displayed by the PacketPresenter. Since each packet is made up of fields, the Fields section defines how the packet is broken up into its fields and what to do with the field data.

### FIELD LINES PROCESSING

---

During processing, the **Fields Section** is processed one **Field Line** at a time in the order that they are listed in the FIELDS section. Each Field Line is parsed against the incoming data packets.

Once a single Field Line is successfully processed and output, the PacketPresenter starts over at the top of the Filed Lines list for the next packet. This ensures that there is only one output packet for each input packet for a given protocol.

There are 2 types of Field Lines. A Field Line can be conditional or unconditional. Unconditional Field Lines are processed for any packet. Conditional Field Lines are only processed if certain fields match a specific value.

Any Unconditional Field Line (no conditionals) generates an output line on the PacketPresenter screen. Any Conditional Field Line that evaluates to True generates an output line on the PacketPresenter screen. Any Conditional Field Line that evaluates to False is skipped and produces no output line on the PacketPresenter screen.

The Field Lines should be listed with the conditional field lines first followed by an unconditional field line to catch all packets that are not explicitly defined in the conditional field lines.

## UNCONDITIONAL FIELD LINES

---

Unconditional Field lines are parsed and decoded data is output for every packet that is input. The Fields specify how to interpret the data and how to output the data.

## CONDITIONAL FIELD LINES

---

Conditional Field Lines provide a means for defining packets whose contents vary based upon the presence of a value in another field. An example of this is a packet that contains a Command Byte that determines the format of the rest of the packet. A Conditional Field Line contains at least one field in the packet that includes the *=Value* token in the input modifiers section.

If the data contained in the conditional fields of a packet matches the *=Value* specified for the field, the packet is parsed and the data is output. If the condition field *=Value* does not match the incoming data, then the processor moves on to the next Field Line until it reaches the end of the Fields section.

## FIELD LINE FORMAT

---

Each Field Line in the Fields Section has the keyword **FIELDS** followed by a series of individual Fields. Individual fields in a packet are separated by commas. A Field line in the Fields Section defines an entire packet from start to end and has the form:

```
Fields  Field1,Field2,. . . ,FieldN
```

You can also insert a string to be printed out at that location in the packet by using the string (\$) operator before the string to be printed. Below is an example of a field line with one string added between the fields.

```
Fields  Field1,$String,. . . ,FieldN
```

Each field will be output with a Label and a Value. For String fields, the Label is blank and the Value is the String.

## FIELD FORMAT

---

Each field in the Field Line is defined using the following syntax and contains no spaces:

```
FieldName.InputModifiers (= value).OutputModifiers
```

**FieldName** is the name of the field. No spaces, commas, semicolons, brackets, dollar signs, periods, or quotes are allowed in the fieldname.

Input and output modifiers change the way incoming data and output data are formatted.

**InputModifiers** are a string of characters that represent how many bits are in the field and how the input data is to be handled. First is the number of bits in the field, or N if the field is a variable length.

Next is any of the following:

- M: native bit order from that which came off of the bus (default)
- L: inverted bit order from that which came off of the bus
- B: invert the Byte order of this multibyte field
- X or Y: which channel the data is on (for multiline busses)
- =Value: Indicates that this field MUST be this value for the entire line to be processed  
(Conditional)

Each modifier is a single character and multiple format modifiers can be combined.

There can only be one Variable length field (using the N input modifier) for each channel (x or y) for any given packet.

Example input modifiers are as follows:

8	; 8 bits
8L	; 8 bits lsb first
8X	; 8 bits from channel X
8Y	; 8 bits from the Y channel
8=47h	; 8 bits that must equal 47 hex. If they do not, then this line is ignored
N	; all bits of the packet (on channel X) not used by other fields
NY	; all bits on channel Y not used by other fields

**OutputModifiers** are a string of characters that represent how to output the contents of this data.

Output Modifiers are as follows:

- I Ignore - no output (entire field is ignored for output)
- D Decimal output
- H Hexadecimal output
- B Binary output
- A Ascii output
- TF True (nonzero) or False (zero)
- TFT True (nonzero) or False (zero), but only show if True
- TFF True (nonzero) or False (zero), but only show if False
- L Look up the text string to print out in a matching Lookup line
- \*Value or /Value: a value to multiply/Divide the output value by
- +Value or -Value: a value to offset the output value by
- \$string: string to print after the data (or in place of the data if the i flag is used). String must be the last item in a field. No commas, quotes, semicolons or parenthesis allowed in the string.

## BUS EVENTS IN THE MIDDLE OF A PACKET

---

Sometimes a specific bus event plays a role in the packet format. To specify that a specific bus event needs to occur at a specific time in the field sequence, place the single Bus Event value inside brackets in the Field Line. Multiple events in a single value are not allowed, however consecutive events are allowed. To indicate the absence of a specific bus event in the protocol, use the ! (Not) operator.

For example, if the bus is I2C, use the following to require that a Start Bit is present between field1 and field2:

```
Fields  Field1, [1], Field2
```

If there is a start bit between the 2 fields, then that Field Line will be processed.

And use the following to require that a Start Bit is NOT present between field1 and field2:

```
Fields  Field1, [!1], Field2
```

If there is a start bit between the 2 fields, then that Field Line will not be processed.

The Bus Events are defined in **Error! Reference source not found.**

## LOOKUP TABLES

---

Often fields contain values that mean something unrelated to the actual number of the data. Lookup Tables provide a way to output a string of text instead of a data value for a field. For each field wanting to use a lookup table, use the “L” output modifier in the field format and then define the table in the FIELDS section using the LOOKUP keyword.

The format of the Lookup table is as follows:

```
LOOKUP Fieldname
[value1]=$string1
[value2]=$string2
. . .
```

*Fieldname* is the name of the field associated with this lookup table. *valuen* refers to the actual data value of the field. *stringn* is the text string that is output instead of the *valuen*.

If a lookup entry is not present for the data value (not found in the Lookup Table or the Lookup Table does not exist), then the data value is output.

For example, the following table will assign the text strings for various values of the data for the CommandByte field. When the field CommandByte,8,L is processed, the strings are output instead of the value

```
Lookup CommandByte
          [0]=$Read
          [1]=$Write
          [2]=$Seek
          [3]=$Loc
          [4]=$Size
```

The Lookup Tables are only associated to the specific Protocol they are contained in. Therefore you can have a CommandByte lookup table in ProtocolA that is different from a CommandByte lookup table in ProtocolB. Within a single Protocol, you need to make sure that the Fieldnames are unique for all Lookup Tables so that the PacketPresenter can determine which table to use.

## EXAMPLES OF FIELD LINES AND FIELDS

---

### JUST PLAIN DATA

---

Fields contain data that may or may not be of interest to the user. Many times the data is information that just needs to be output to the viewer. Being binary data, each field may need to be translated numerically to mean something. To output a field of data, you can specify the radix (if it should be shown in Hex, Decimal, binary) as well as a gain and offset to scale the data. Finally you can add a string to the field to complete the information. All scaling is performed first using floating point and then the output formatting is applied.

Below is an example of a field to just output the data.

```
Fields Volts.16m.d*1.5-37.256$mV
```

This Field Line contains one field named “Volts”, which is 16 bits long in msbit first order. The output is to be displayed in decimal format, multiplied by 1.5, offset by - 37.256 and finally appended with “mV” before output to the PacketPresenter screen.

For an input packet as follows:

```
0000001100001100. . .
```

The output would be:

Volts
1132.744mV

which is the input 16 bits in msbfirst order (0x30C) times the gain of 1.5 plus the offset of -37.256 output in decimal format plus the “mV” string.

### CONDITIONAL PACKET FORMAT

---

Using the Conditional input modifier, many different field arrangements can be defined for the same packet. Common uses are for parameter fields that exist for different types of commands. If packets contain commands that determine what the remaining fields are, this syntax defines what those remaining fields are.

Below is an example of various packet formats based on a single command field.

```
Fields Command.4m=0.h,Address.8m.h
Fields Command.4m=2.h,Address.8m.h,Data.8m.h
Fields Command.4m=4.h,Param1.8m.h,Param2.8m.h,Param3.8m.h
```

For an input packet as follows:

0010 00011101 00001000. . .

Followed by a packet:

0100 00011101 00001000 11111110. . .

The output would be:

Command	Address	Data
2	1D	08

Command	Param1	Param2	Param3
4	1D	08	FE

which are the fields associated with the Command=2 and Command=4 Field Lines.

## STRING LOOKUP

---

Fields that can be better expressed as text strings can be outputted as such using a Lookup table.

Below is an example of a field that uses a lookup table.

```
[Fields]
Fields StartByte.8.H, CommandByte.8.L, EndByte.8.H
Lookup CommandByte
[0]=$Read
[1]=$Write
[2]=$Seek
[3]=$Loc
[4]=$Size
```

For an input packet as follows:

00100001 00000001 00001000. . .

The output would be:

StartByte	Command	EndByte
21	Write	08

which is the text associated with the Command Field 4 bits in msbfirst order (0010b = 2).

## CONDITIONAL ROUTE OF DATA TO ANOTHER PROTOCOL

---

Many embedded protocols support multiple layers of protocol, where each protocol layer handles a different set of services or functions. In these multilayer protocols, a field of data from one protocol layer may be the input data to another layer of protocol. Routing this field of data to a new Protocol is as easy as naming the Field the same name as the Protocol. If the Field name matches any protocol, the entire data for that field is passed to that Protocol for processing.



Below is an example that shows a field being sent to a new layer (Layer2) of protocol when the command field is a 1.

```
[Protocol]
name = Layer1
[Packet]
[Decode]
[Fields]
Fields Command.4=0.h,Address.8.h
Fields Command.4=1.h,Layer2.48.h

[Protocol]
name = Layer2
[Packet]
[Decode]
[Fields]
Fields L2Command.4=0.h,RSSI.8.d
Fields L2Command.4=1.h,QoS.16.d
Fields L2Command.4=2.h,Layer3.44.h
```

---

## PACKETPRESENTER ADD-IN API

The USBee RX PacketPresenter automatically processes many types of data streams. However, it cannot decode custom coded data streams. Using the PacketPresenter Add-In API, the data stream can be decoded to the basic data values for any custom coding.

The USBee RX software package includes a sample DLL project in Microsoft VC6 format (in the installation directory of the USBee RX software) called AddIn that allows you to customize a decoder for your data streams.

The DLL library called usbeeai.dll (USBee Add-In) has the following interface routine that is called by the PacketPresenter if the ADDIN keyword is used in the DECODE section of the PacketPresenter Definition File.

```
CWAV_EXPORT unsigned int CWAV_API  APIDecode(  
                                char *Protocol,  
                                char bitIn,  
                                char &bitOut,  
                                char reset );
```

This routine is called for each bit of data in the data stream. Protocol is the string name of the Protocol being processed and allows you to create an add-in that handles many different kinds of decoding. The parameter “reset” is set to a 1 for the first bit of a packet and 0 for all bits following. The next bit from the stream is passed in using the parameter “bitIn” (1 or 0).

After your code decodes the stream, you can either send back no data (return value of 0), or send a new bits back using the “bitOut” pointer (one bit per char) and a return value of the number of bits returned.

The default Add-In routine simply is a pass through so that the output data stream equals the input data stream. Start with this library source code to add your custom decoding.

---

## SAMPLE PACKETPRESENTER ADD-IN DECODERS

Custom decoders can perform complicated decryption and byte or bit manipulation. Ignoring the actual algorithm that is executed, these decoders may reduce, enlarge or keep constant the number of bits in the data stream. The following examples are intended to show how these streams can be shortened, lengthened or modified. Useful decoders will need to have the appropriate algorithms to compute the true values of the output bits.

### LOOPBACK DECODER

---

This Add-In simply loops back the data (out = in).

```
CWAV_EXPORT unsigned int CWAV_API APIDecode(char *Protocol, char bitIn, char *bitsOut, char reset )
{
    // This will be the Add-In routine that is called by the PacketPresenter
    // when the ADDIN keyword is used in the DECODE section of the
    // PacketPresenter Definition File.

    // This routine is called for each bit of data in a data packet.
    // The parameter "reset" is set to a 1 for the first bit of a packet and
    // 0 for all bits following. The next bit from the stream is passed in
    // using the parameter "bitIn" (1 or 0). After your code decodes the stream,
    // you can either send back no data (return value of 0), or send new bits back
    // using the "bitOut" pointer (one bit per char) and a return value of the number
    // of bits returned. The default Add-In routine is simply a pass through so
    // that the output data stream equals the input data stream.
    // Start with this library source code to add your custom decoding.

    *bitsOut = bitIn;

    return( 1 );           // Indicates that there is 1 return data bit
}
```

### INVERTING DECODER

---

This Add-In inverts the packet data (out = Not(in)).

```
CWAV_EXPORT unsigned int CWAV_API APIDecode(char *Protocol, char bitIn, char *bitsOut, char
reset )
{
    if (bitIn)
        *bitsOut = 0;
    else
        *bitsOut = 1;
    return( 1 );           // Indicates that there is 1 return data bit
}
```

### EXPANDING DECODER

---

This Add-In shows how to convert a stream to a larger stream (expanding the bits). In this case each bit becomes two output bits.

```
CWAV_EXPORT unsigned int CWAV_API APIDecode(char *Protocol, char bitIn, char *bitsOut,
char reset )
{
    *bitsOut++ = bitIn;
    *bitsOut++ = bitIn;

    return( 2 );           // Indicates that there is 2 return data bits
}
```

## COMPRESSING DECODER

---

This Add-In shows how to remove bits from a stream (compressing the bits). In this case each bit pair becomes a single bit, basically throwing away the first bit.

```
CWAV_EXPORT unsigned int CWAV_API APIDecode(char *Protocol, char bitIn, char *bitsOut,
char reset )
{
    static everyother = 0;

    if (reset)                                // Reset the state of the decoder if
reset=TRUE
        everyother = 0;

    if (everyother)
    {
        *bitsOut = bitIn;
        return( 1 );                          // Indicates that there is 1 return data
bit
    }
    everyother = 0;
    else
        everyother = 1;

    return( 0 );                              // Indicates that there are no return data bits
}
```

## MULTIPLE DECODERS

---

This Add-In shows how to use the Protocol string to selectively decode different types of packets.

```
CWAV_EXPORT unsigned int CWAV_API APIDecode(char *Protocol, char bitIn, char *bitsOut,
char reset )
{
    static everyother = 0;

    if (!strcmp( Protocol, "COMPRESS")
    {
        if (reset)                // Reset the state of the decoder if reset=TRUE
            everyother = 0;
        if (everyother)
        {
            *bitsOut = bitIn;
            return( 1 );          // Indicates that there is 1 return data bit
            everyother = 0;
        }
        else
            everyother = 1;
        return( 0 );              // Indicates that there are no return data bits
    }
    else if (!strcmp( Protocol, "EXPAND")
    {
        *bitsOut++ = bitIn;
        *bitsOut++ = bitIn;
        return( 2 );              // Indicates that there is 2 return data bits
    }

    // No matching decoder label found so just loopback the data
    *bitsOut = bitIn;

    return(1);
}
```

---

## PACKETPRESENTER DEFINITION FILE DEBUGGING

Creating your PacketPresenter Definition File can be made simpler using the Debug mode. To turn on Debug mode, use the DebugOn keyword in **ALL** [DEBUG] sections of the Definition File.

```
[Protocol]
                                name = I2CEEPROM

[DEBUG]
                                DebugOn           ; Turns On Debug Mode.
                                                ; Comment it out to turn it off.

[Packet]
```

When debug mode is on, each packet is output twice in its raw form, showing the data values as well as the events from the bus. The first debug line is the initial bus data. The second line is the bus data after any decoding is completed. Following the debug lines are the PacketPresenter output packets from this same data.

Below is a screen shot that shows the PacketPresenter that has Debug turned on.



## PACKETPRESENTER SPECIFICATIONS

The PacketPresenter system has the following limits regarding file size, packets, fields, lookup tables etc.

- 100K bytes per PacketPresenter Definition File
- 64K Data Records per Packet (min 64K bits, max 64K bytes)
- 7 Protocols
- 1024 Field Lines per Protocol
- 128 Fields per Field Line
- 64 Lookup Tables per Protocol
- 256 Lookup entries per Lookup Table
- 256 Decoder Substitutions per Protocol
- 3 Bytes per Substitution input or output
- 4 PacketPresenter Windows
- 2.1B bytes per PacketPresenter Output File

---

### ASYNCR PROTOCOL EXAMPLE

```
; Async Protocol Definition File
; This file defines the transfers to/from a custom device
; over an ASYNCR bus
;
[Protocol]
    name = ASYNCRBus
    bitwise
[DEBUG]
    ;DebugOn          ; Uncomment this to turn on Debug Packets
[Packet]
    [Start]
        type = value
        value = 40h; Start command
        mask = F0h ; Mask out the channel number

    [End]
        type = timeout
        timeout = 3000 ; 3ms timeout ends the packet

[Decode]
[Fields]

    Fields
        Start.4.h,
        Channel.4=1.h,
        Command.8.h,
        X.16.d/20.48-25$g,
        Y.16.d/20.48-25$g,
        Z.16.d/20.48-25$g,
        Rest.N.h    ; Rest of the packet

    Fields
        Rest.N.h    ; Rest of the packet
```

## I2C PROTOCOL EXAMPLE

```
; I2C EEPROM Protocol Definition File
; This file defines the transfers to/from an I2C EEPROM
; with 8 bit address
;
[Protocol]
    name = I2CEEPROM
    bitwise
[DEBUG]
; DebugOn          ; Uncomment this to turn on Debug Packets
[Packet]
[Start]
    type = event
    event = 1 ; Start Bit

[End]
    type = event
    event = 0Ah ; Stop Bit Or NACK

[Decode]
[Fields]

; Device Not Present
Fields
    $Device Not Present,          ; Printout this label if match
    SlaveAddress.7m.h,RW.1.i,    ; Control Byte
    Address.8m.h,                ; 1 byte address
    [8]                          ; followed by a NACK condition

; Set Address
Fields
    $SetAddressCmd,              ; Printout this label if match
    SlaveAddress.7m.h,RW.1=0.i,  ; Control Byte
    Address.8m.h,                ; 1 byte address
    [2]                          ; followed by a STOP condition

; Write Command
Fields
    $WriteCommand,              ; Printout this label if match
    SlaveAddress.7m.h,RW.1=0.i,  ; Control Byte
    Address.8m.h,                ; 1 byte address
    [!1],                        ; NO START condition
    WriteData.Nm.h               ; Written Data (Variable N)

; Current Address Read
Fields
    $CurrentRead,               ; Printout this label if match
    SlaveAddress.7m.h,RW.1=1.i,  ; Control Byte
    ReadData.Nm.h               ; Read Data (Variable number N)

; Random Read
Fields
    $RandomRead,                ; Printout this label if match
    SlaveAddress.7m.h,RW.1=0.i,  ; Control Byte
    Address.8m.h,                ; 1 byte address
    [1],                         ; START Condition
    SlaveAddress.7m.i,RW.1=1.i,  ; Control Byte
    ReadData.Nm.h,              ; Read Data (Variable number N)
```



## SPI PROTOCOL EXAMPLE

```
; Cypress RF IC Protocol Definition File
; This file defines the transfers to/from a CY6936 RF IC
; using the SPI bus
[Protocol]
    name = CypressRFIC
    bitwise
[DEBUG]
    ;DebugOn
[Packet]
    [Start]
        type = event
        event = 1 ; SS goes active

    [End]
        type = event
        event = 2 ; SS goes inactive
[Decode]
[Fields]
    ; RX_IRQ_STATUS_ADR Read and Write Command
    Fields   Dir.ly=0.L, Inc.ly.tf, Address.6y=07h.L, Dummy.8x.i, RXOW.1x.h,
            SOPDET.1x.h, RXB16.1x.h, RXB8.1x.h, RXB1.1x.h, RXBERR.1x.h, RxC.1x.h,
            RXE.1x.h
    Fields   Dir.ly=1.L, Inc.ly.tf, Address.6y=07h.L, RXOW.1y.h, SOPDET.1y.h,
            RXB16.1y.h, RXB8.1y.h, RXB1.1y.h, RXBERR.1y.h, RxC.1y.h, RXE.1y.h

    ; TX_IRQ_STATUS_ADR Read and Write Command
    Fields   Dir.ly=0.L, Inc.ly.tf, Address.6y=04h.L, Dummy.8x.i, OS.1x.h, LV.1x.h,
            TXB15.1x.h, TXB8.1x.h, TXB1.1x.h, TXBERR.1x.h, TXC.1x.h, TXE.1x.h
    Fields   Dir.ly=1.L, Inc.ly.tf, Address.6y=04h.L, OS.1y.h, LV.1y.h, TXB15.1y.h,
            TXB8.1y.h, TXB1.1y.h, TXBERR.1y.h, TXC.1y.h, TXE.1y.h

    ; RX_BUFFER_ADR Read and Write Command
    Fields   Dir.ly=0.L, Inc.ly.tf, Address.6y=21h.L, Dummy.8x.i,
            RxData.Nx.h
    Fields   Dir.ly=1.L, Inc.ly.tf, Address.6y=21h.L, RxData.Ny.h

    ; TX_BUFFER_ADR Read and Write Command
    Fields   Dir.ly=0.L, Inc.ly.tf, Address.6y=20h.L, Dummy.8x.i,
            TxData.Nx.h
    Fields   Dir.ly=1.L, Inc.ly.tf, Address.6y=20h.L, TxData.Ny.h

    Fields   Dir.ly=0.L, Inc.ly.tf, Address.6y.L, Dummy.8x.i,
            ReadData.Nx.h
    Fields   Dir.ly=1.L, Inc.ly.tf, Address.6y.L, WriteData.Nmy.h

Lookup Dir
    [0]=$Read
    [1]=$Write

Lookup Address
    [00h]=$CHANNEL_ADR
    [01h]=$TX_LENGTH_ADR
    [02h]=$TX_CTRL_ADR
    [03h]=$TX_CFG_ADR
    [04h]=$TX_IRQ_STATUS_ADR
    [05h]=$SRX_CTRL_ADR
    [06h]=$SRX_CFG_ADR
    [07h]=$SRX_IRQ_STATUS_ADR
    [08h]=$SRX_STATUS_ADR
    [09h]=$SRX_COUNT_ADR
    [0ah]=$SRX_LENGTH_ADR
    [0bh]=$SPWR_CTRL_ADR
    [0ch]=$XTAL_CTRL_ADR
    [0dh]=$IO_CFG_ADR
    [0eh]=$GPIO_CTRL_ADR
    [0fh]=$XACT_CFG_ADR
    [10h]=$FRAMING_CFG_ADR
    [11h]=$DATA32_THOLD_ADR
    [12h]=$DATA64_THOLD_ADR
    [13h]=$RSSI_ADR
    [14h]=$EOP_CTRL_ADR
    [15h]=$CRC_SEED_LSB_ADR
    [16h]=$CRC_SEED_MSB_ADR
    [17h]=$TX_CRC_LSB_ADR
    [18h]=$TX_CRC_MSB_ADR
```

```

[19h]=$RX_CRC_LSB_ADR
[1ah]=$RX_CRC_MSB_ADR
[1bh]=$TX_OFFSET_LSB_ADR
[1ch]=$TX_OFFSET_MSB_ADR
[1dh]=$MODE_OVERRIDE_ADR
[1eh]=$RX_OVERRIDE_ADR
[1fh]=$TX_OVERRIDE_ADR
[26h]=$XTAL_CFG_ADR
[27h]=$CLK_OVERRIDE_ADR
[28h]=$CLK_EN_ADR
[29h]=$RX_ABORT_ADR
[32h]=$AUTO_CAL_TIME_ADR
[35h]=$AUTO_CAL_OFFSET_ADR
[39h]=$ANALOG_CTRL_ADR
[20h]=$TX_BUFFER_ADR
[21h]=$RX_BUFFER_ADR
[22h]=$SOP_CODE_ADR
[23h]=$DATA_CODE_ADR
[24h]=$PREAMBLE_ADR
[25h]=$MFG_ID_ADR

[Protocol]
    name = RxData
    bitwise
[DEBUG]
    ;DebugOn
[Packet]
    [Start]
        type = next
    [End]
        type = event
        event = 127      ; All Data passed in

[Decode]
[Fields]
    ; RX_IRQ_STATUS_ADR Read and Write Command
    Fields      ReceiveData.N.h

```

## CAN PROTOCOL EXAMPLE

```
; CAN Protocol Definition File
; This file defines the transfers to/from a custom CAN device
; over a the CAN bus
;
[Protocol]
    name = CANBus
    bitwise

[DEBUG]
    ; DebugOn          ; Uncomment this to turn on Debug Packets

[Packet]
    [Start]
        type = event
        event = 1 ; Start of CAN packet

    [End]
        type = event
        event = 2 ; End of CAN packet

[Decode]
[Fields]

    ; Extended Frame Format
    Fields    SOF.1.i, IDA.11.h, SRR.1.h, IDE.1=1.h, IDB.18.h, RTR.1.h,
              Rsr.2.i, Length.4.h, Data.N.h, CRC.15.h, CRCDel.1.h,
              ACK.1.h, ACKDel.1.h, EOF.7.h

    ; Base frame format
    Fields    SOF.1.i, ID.11.h, RTR.1.h, IDE.1=0.h, Rsr.1.i, Length.4.h,
              Data.N.h, CRC.15.h, CRCDel.1.h, ACK.1.h, ACKDel.1.h,
              EOF.7.h
```

## SDIO PROTOCOL EXAMPLE

```
; SDIO Protocol Definition File
; This file defines the transfers to/from a custom device
[Protocol]
name = SDIO_0
bitwise
[Debug]
; DebugOn          ; Uncomment this to turn on Debug Packets
[Packet]
    [Start]
        type = event
        event = 1 ; Start of Packet
        channelxory

    [End]
        type = event
        event = 2 ; End of Packet
        channelxory

[Decode]
[Fields]

    ; Data Packets
    Fields [8], Data.Ny.h

    ; Do all of the Command Packets
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=4.L, DSR.16x.h
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=7.L, RCA.16x.h
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=9.L, RCA.16x.h
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=10.L, RCA.16x.h
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=13.L, RCA.16x.h
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=15.L, RCA.16x.h
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=16.L, BLOCK_LEN.32x.h
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=17.L, Address.32x.h
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=18.L, Address.32x.h
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=24.L, Address.32x.h
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=25.L, Address.32x.h
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=28.L, Address.32x.h
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=29.L, Address.32x.h
    Fields [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=30.L, Address.32x.h
```

```

Fields      [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=32.L, Address.32x.h
Fields      [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=33.L, Address.32x.h
Fields      [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=55.L, RCA.16x.h
Fields      [!3], Start.1x.h, HOST.1x=1.h, CMD.6x=56.L, nua.31x.i, RD/WR#.1.h
; All of the App Command packets
Fields      [3], Start.1x.h, HOST.1x=1.h, ACMD.6x=6.L, nua.30x.i, BUS_WIDTH.2.h
Fields      [3], Start.1x.h, HOST.1x=1.h, ACMD.6x=13.L
Fields      [3], Start.1x.h, HOST.1x=1.h, ACMD.6x=23.L, nua.9x.i, BLOCKS.23.h
Fields      [3], Start.1x.h, HOST.1x=1.h, ACMD.6x=41.L, OCR.32.h
Fields      [3], Start.1x.h, HOST.1x=1.h, ACMD.6x=42.L, nua.30x.i, SET_CD.1.h
Fields      [3], Start.1x.h, HOST.1x=1.h, ACMD.6x.L
Fields      [!3], Start.1x.h, HOST.1x=1.h, CMD.6x.L
; The R6 response packet
Fields      [7], $R6, Start.1x.h, DEVICE.1x=0.h, CMD.6x.L, RCA.16x.h, STATUS.16x.h
; The R3 response packet
Fields      [6], $R3, Start.1x.h, DEVICE.1x=0.h, CMD.6x.L, OCR.32x.h
; The R2 response packet
Fields      [5], $R2, Start.1x.h, DEVICE.1x=0.h, CMD.6x.L, DATA.128x.h
; The R1 response packet
Fields      [4], $R1, Start.1x.h, DEVICE.1x=0.h, CMD.6x.L,
    OUT_OF_RANGE.1x.tft,
    ADDRESS_ERROR.1x.tft,
    BLOCK_LEN_ERROR.1x.tft,
    ERASE_SEQ_ERROR.1x.tft,
    ERASE_PARAM.1x.tft,
    WP_VIOLATION.1x.tft,
    CARD_IS_LOCKED.1x.tft,
    LOCK_UNLOCK_FAILED.1x.tft,
    COM_CRC_ERROR.1x.tft,
    ILLEGAL_COMMAND.1x.tft,
    CARD_ECC_FAILED.1x.tft,
    CC_ERROR.1x.tft,
    ERROR.1x.tft,
    UNDERRUN.1x.tft,
    OVERRUN.1x.tft,
    CID/CSD_OVERWRITE.1x.tft,
    WP_ERASE_SKIP.1x.tft,
    CARD_ECC_DISABLE.1x.tft,
    ERASE_REST.1x.tft,
    CURRENT_STATE.4x.L,
    READY_FOR_DATA.1x.tff,
    nul.2x.i,
    APP_CMD.1x.tft,
    nu4.1x.i,
    AKE_SEQ_ERROR.1x.tft,
    RSV_2.1x.i,
    RSV_TEST_MODE.2x.i

```

Lookup CURRENT\_STATE

```

[0]=$Idle
[1]=$Ready
[2]=$Ident
[3]=$Stby
[4]=$Tran
[5]=$Data
[6]=$Rcv
[7]=$Prg
[8]=$Dis
[9]=$reserved
[10]=$reserved
[11]=$reserved
[12]=$reserved
[13]=$reserved
[14]=$reserved
[15]=$reserved

```

Lookup CMD

```

[0]=$GO_IDLE_STATE
[1]=$CMD1
[2]=$ALL_SEND_CID
[3]=$SEND_RELATIVE_ADDR
[4]=$SET_DSR

```

```

[5]=$CMD5
[6]=$SWITCH_FUNC
[7]=$SEL/DESEL_CARD
[8]=$SEND_IF_COND
[9]=$SEND_CSD
[10]=$SEND_CID
[11]=$VOLTAGE_SWITCH
[12]=$STOP_TRANSMISSION
[13]=$SEND_STATUS
[14]=$CMD14
[15]=$GO_INACTIVE_STATE
[16]=$SET_BLOCKLEN
[17]=$READ_SINGLE_BLOCK
[18]=$READ_MULTIPLE_BLOCK
[19]=$SEND_TUNING_BLOCK
[20]=$SPEED_CLASS_CONTROL
[21]=$CMD21
[22]=$CMD22
[23]=$SET_BLOCK_COUNT
[24]=$WRITE_BLOCK
[25]=$WRITE_MULTIPLE_BLOCK
[26]=$CMD26
[27]=$PROGRAM_CSD
[28]=$SET_WRITE_PROT
[29]=$CLR_WRITE_PROT
[30]=$SEND_WRITE_PROT
[31]=$CMD31
[32]=$ERASE_WR_BLK_START
[33]=$ERASE_WR_BLK_END
[34]=$CMD34
[35]=$CMD35
[36]=$CMD36
[37]=$CMD37
[38]=$ERASE
[39]=$CMD39
[40]=$CMD40
[41]=$CMD41
[42]=$LOCK_UNLOCK
[43]=$CMD43
[44]=$CMD44
[45]=$CMD45
[46]=$CMD46
[47]=$CMD47
[48]=$CMD48
[49]=$CMD49
[50]=$CMD50
[51]=$CMD51
[52]=$CMD52
[53]=$CMD53
[54]=$CMD54
[55]=$APP_CMD
[56]=$GEN_CMD
[57]=$CMD57
[58]=$CMD58
[59]=$CMD59
[60]=$CMD60
[61]=$CMD61
[62]=$CMD62
[63]=$CMD63
Lookup ACMD
[6]=$SET_BUS_WIDTH
[13]=$SD_STATUS
[22]=$SEND_NUM_WR_BLOCKS
[23]=$SET_WR_BLK_ERASE_COUNT
[41]=$SD_APP_OP_COND
[42]=$SET_CLR_CARD_DETECT
[51]=$SEND_SCR

```

## 1-WIRE PROTOCOL EXAMPLE

```
; One Wire Protocol Definition File
; This file defines the transfers to/from some 1-Wire devices
; using the 1-Wire bus
;
[Protocol]
    name = OneWireBus
    bitwise

[DEBUG]
; DebugOn                      ; Uncomment this to turn on Debug Packets

[Packet]
[Start]
    type = event
    event = 2 ; Presence Pulse

[End]
    type = event
    event = 1 ; Reset Pulse

[Decode]
[Fields]

; These fields are used by Maxim/Dallas Digital Thermometers
Fields    ROMCommand.8=F0h.$Search Rom, Data.N.h
Fields    ROMCommand.8=33h.$Read Rom, Family.8.h, SerialNumber.48.h,
          CRC.8.h
Fields    ROMCommand.8=55h.$Match Rom, Family.8.h, SerialNumber.48.h, CRC.8.h
Fields    ROMCommand.8=CCh.$Skip ROM, Function.8=44h.$ConvertTemp
Fields    ROMCommand.8=CCh.$Skip ROM, Function.8=BEh.$Read Scratchpad, Temp.16.d,
          TH.8.h, TL.8.l.h, Rsvd.16.i, Remain.8.h, CpC.8.h, CRC.8.h

; These fields are used by Dallas Serial Number iButtons
Fields    ROMCommand.8=33h.$Read Rom, Family.8.h, SerialNumber.48.h, CRC.8.h
Fields    ROMCommand.8=0Fh.$Read Rom, Family.8.h, SerialNumber.48.h, CRC.8.h

; These packets are used by 1-Wire EEPROMS
Fields    ROMCommand.8=33h.$Read Rom, Family.8.h, SerialNumber.48.h, CRC.8.h
Fields    ROMCommand.8=0Fh.$Write Scratchpad,
          Address.16.h, Data.N.h
Fields    ROMCommand.8.h, MemoryCommand.8=AAh.$Read Scratchpad,
          Address.16.h, ES.8.h, Data.N.h
Fields    ROMCommand.8.h, MemoryCommand.8=55h.$Copy Scratchpad,
          AuthCode.24.h
Fields    ROMCommand.8.h, MemoryCommand.8=F0h.$Read Memory,
          Address.16.h, Data.N.h
```

## PARALLEL PROTOCOL EXAMPLE

```
; Sample Parallel Protocol Definition File
; This file defines the transfers to/from an unique device
;
[Protocol]
    name = ADevice
    bitwise
[DEBUG]
    ;DebugOn
[Packet]
    [Start]
        type = signal
        signal = 14
        level = 0

    [End]
        type = length
        Bytelength = 21

[Decode]
[Fields]
    Fields
        StartByte.8m.d*2+4$mV,
        CommandByte.8l.L,
        FLength.8m.h,
        SlaveAddress.7m.h,RW.1.L,
        Long.32m.h,
        8Bytes.64m.h,
        NextLayer.Nm.h

[Protocol]
    name = NextLayer
    bitwise
[Packet]
    [Start]
        type = next
    [End]
        type = Event    ;End of a packet is signaled by a event
        event = 127; Means the end of the data (only for higher
layers)

[Decode]
[Fields]
    Fields
        Rest.N.h        ; Just print out all the bytes
```

## SERIAL PROTOCOL EXAMPLE

```
; Serial Protocol Definition File
; This file defines the transfers from a serial device
;
[Protocol]
    name = SerialBus
    bitwise
[DEBUG]
    ;DebugOn          ; Uncomment this to turn on Debug Packets
[Packet]
    [Start]
        type = value    ; Look for a value in the data to start the
packet
        value = 6211h    ; NOTE: This value is assumed MSbit first in
                           ; the data stream!
        bits = 16
        mask = FFFFh
    [End]
        type = length
        bitlength = 64 ; End of command after 64 bits
[Decode]
[Fields]

    ; Send out the bits of the packet
    Fields Start.16.h, Nine.9.h, Seven.7.h, Rest.N.b
```



## USB PROTOCOL EXAMPLE

```
; USB Bus Protocol Definition File
; This file defines the transfers to/from a custom USB device
;
[Protocol]
    name = USBBus
    bitwise
[DEBUG]
    ;DebugOn          ; Uncomment this to turn on Debug Packets
[Packet]
    [Start]
        type = event
        event = 1 ; Setup/In or Out found
    [End]
        type = event
        event = 6 ; ACK, NAK or Stall found or no handshake found
[Decode]
[Fields]

; Any Packet - No Response
Fields    Sync.8.i, PID.8.L, Addr.7L.d, EP.4L.d, CRC5.5.i, ; Token
          [4]                                           ; No Handshake

; Setup - Nakd
Fields    Sync.8.i, PID.8=10110100b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, HS.8=01011010b.L                    ; Handshake

; IN - Nakd
Fields    Sync.8.i, PID.8=10010110b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, HS.8=01011010b.L                    ; Handshake

; OUT - Nakd
Fields    Sync.8.i, PID.8=10000111b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, HS.8=01011010b.L                    ; Handshake

; Setup
Fields    Sync.8.i, PID.8=10110100b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, PID.8.L, Rtype.8.i,
          bRequest.8L=1.$Clear Feature, bValue.16L.h, bIndex.16L.H,
          bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L

Fields    Sync.8.i, PID.8=10110100b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, PID.8.L, Rtype.8.i,
          bRequest.8L=0.$Get Status, bValue.16L.h, bIndex.16L.H,
          bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L

Fields    Sync.8.i, PID.8=10110100b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, PID.8.L, Rtype.8.i,
          bRequest.8L=8.$Get Configuration, bValue.16L.h, bIndex.16L.H,
          bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L

Fields    Sync.8.i, PID.8=10110100b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, PID.8.L, Rtype.8.i,
          bRequest.8L=6.$Get Descriptor, bValue.8L.I, Type.8L.L,
          bIndex.16L.H, bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L

Fields    Sync.8.i, PID.8=10110100b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, PID.8.L, Rtype.8.i,
          bRequest.8L=16.$Get Interface, bValue.16L.h, bIndex.16L.H,
          bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L

Fields    Sync.8.i, PID.8=10110100b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, PID.8.L, Rtype.8.i,
          bRequest.8L=5.$Set Address, Address.16L.h, bLength.16L.i,
          bLength.16L.i, CRC16.16.i, Sync.8.i, HS.8.L

Fields    Sync.8.i, PID.8=10110100b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, PID.8.L, Rtype.8.i,
          bRequest.8L=9.$Set Configuration, Config.16L.h,
          bLength.16L.i, bLength.16L.i, CRC16.16.i, Sync.8.i, HS.8.L

Fields    Sync.8.i, PID.8=10110100b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, PID.8.L, Rtype.8.i,
          bRequest.8L=7.$Set Descriptor, bValue.16L.h, bIndex.16L.H,
          bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L

Fields    Sync.8.i, PID.8=10110100b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, PID.8.L, Rtype.8.i,
          bRequest.8L=3.$Set Feature, bValue.16L.h, bIndex.16L.H,
          bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L
```

```

Fields      Sync.8.i, PID.8=10110100b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
            Sync.8.i, PID.8.L, Rtype.8.i,
            bRequest.8L=10.$Get Interface, bValue.16L.h, bIndex.16L.H,
            bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L
Fields      Sync.8.i, PID.8=10110100b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
            Sync.8.i, PID.8.L, Rtype.8.i,
            bRequest.8L=11.$Set Interface, AltSetting.16L.h,
            Interface.16L.H, bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L
Fields      Sync.8.i, PID.8=10110100b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
            Sync.8.i, PID.8.L, Rtype.8.i,
            bRequest.8L=12.$Sync Frame, bValue.16L.h, bIndex.16L.H,
            bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L

; IN
Fields      Sync.8.i, PID.8=10010110b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
            Sync.8.i, PID.8.L, InData.NL.h, CRC16.16.i,      ; Data
            Sync.8.i, HS.8.L                                ; Handshake

; OUT
Fields      Sync.8.i, PID.8=10000111b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
            Sync.8.i, PID.8.L, OutData.NL.h, CRC16.16.i,    ; Data
            Sync.8.i, HS.8.L                                ; Handshake

; Catch all
Fields Data.NL.h

Lookup      Type
            [1]=$Device
            [2]=$Config
            [3]=$String

Lookup      PID
            [11000011b]=$DATA0
            [11010010b]=$DATA1
            [01001011b]=$ACK
            [01011010b]=$NAK
            [01111000b]=$STALL
            [10110100b]=$SETUP
            [10000111b]=$OUT
            [10010110b]=$IN
            [10100101b]=$SOF

Lookup      HS
            [01001011b]=$ACK
            [01011010b]=$NAK
            [01111000b]=$STALL

```

## PS2 PROTOCOL EXAMPLE

```
; PS2 Protocol Definition File
; This file defines the transfers from a PS2 device
;
[Protocol]
    name = PS2Bus
    bitwise
[DEBUG]
    ;DebugOn          ; Uncomment this to turn on Debug Packets
[Packet]
    [Start]
        type = next      ; Every byte is the start of the next packet
        CHANNELXORY      ; Either Device to Host or Host To Device
    [End]
        type = TIMEOUT
        TIMEOUT = 5000 ; End of command after 5msec
[Decode]
[Fields]

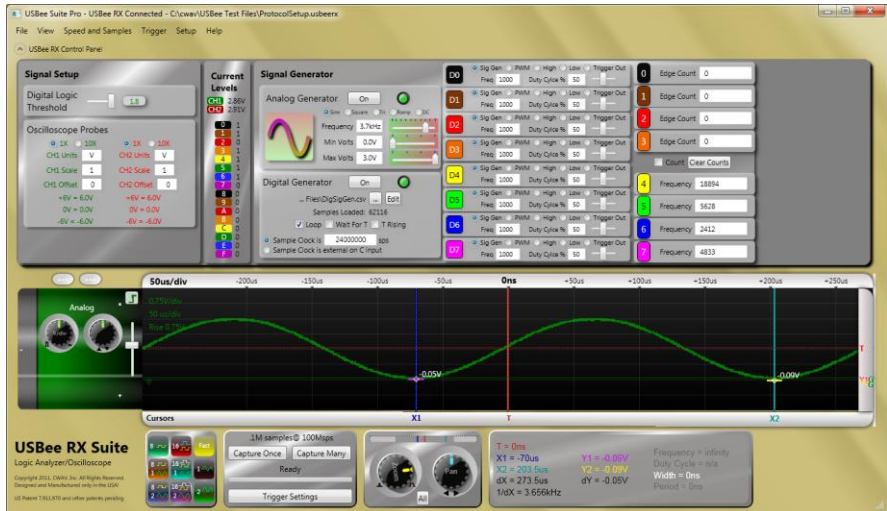
; Setting LEDs after command
Fields [1], $Device To Host, $Key Down, Scancode.8x.h, [2],
        $Host To Device, HostCommand.8y=EDh.$Set LEDs,
        Ack.8x.i, Parameter.5y.i, Caps.1y.tf, Num.1y.tf,
        Scroll.1y.tf, Ack.8x.i
Fields [1], $Device To Host, $Key Down, Scancode.8x.h, [2],
        $Host To Device, HostCommand.8y.h, Ack.8x.i,
        Parameter.8y.h, Ack.8x.i

; Device to Host
Fields [1], $Device To Host, $Key Up, Release.8x=F0h.h,
        Scancode.Nx.h

; All other scancodes
Fields [1], $Device To Host, $Key Down, Scancode.Nx.H

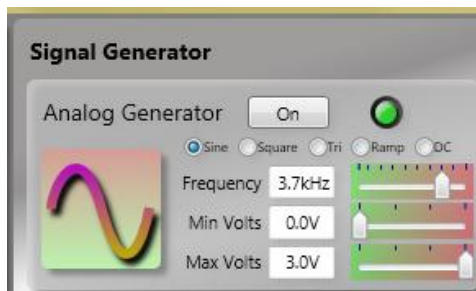
; Host to Device
Fields [2], $Host To Device, Command.Ny.h
```

# USBEE RX ANALOG SIGNAL GENERATOR



The USBee RX has an Analog Signal Generator built in. It outputs a voltage pattern on the **Aout** signal out the side of the Pod.

To specify what waveform pattern is generated, you use the Analog Generator section of the USBee RX Control Panel as seen below.



There are 5 types of analog signals that can be generated by selecting the associated radio button:

- Sine
- Triangle
- Ramp
- Square and
- DC

You can also select the frequency generated by using the slider to the right of the Frequency line. The minimum and maximum voltages are also specified using the sliders from 0V to 3.0V.

Below are a few of the waveforms available.





The USBee RX constantly monitors the state of all of the digital inputs and the voltage levels of the CH1 and CH2 inputs. This way you can see the state of the channels even at times you are not capturing waveforms using the Mixed Signal Oscilloscope.

These levels are updated every 500msecs.

## USBEE DIGITAL LOGIC THRESHOLD



The USBee RX features variable Digital Logic Thresholds that can range from -1V to +2V. Set this logic threshold to define the voltage level that indicates the change from a logic "0" to a logic "1".

If this level is set incorrectly, or too close to either the top or bottom of your logic range, you will see inconsistent waveforms and this level will need to be adjusted.

## USBEE RX DIGITAL SIGNAL GENERATOR

The USBee RX has an Digital Signal Generator built in. It outputs a digital voltage pattern on the **D0** through **D7** signals out the side of the Pod.

To specify what waveform pattern is generated on each of the output signals,, you use the section of the USBee RX Control Panel as seen below:



Each of the 8 digital lines can have one of 5 different modes by selecting the associated radio button above:

- The output of the Arbitrary Digital Signal Generator
- Pulse Width Modulator
- Logic High Level
- Logic Low Level
- Trigger Out signal

The Logic **Low** level outputs 0V on the signal.

The Logic **High** level outputs 3.3V on the signal.

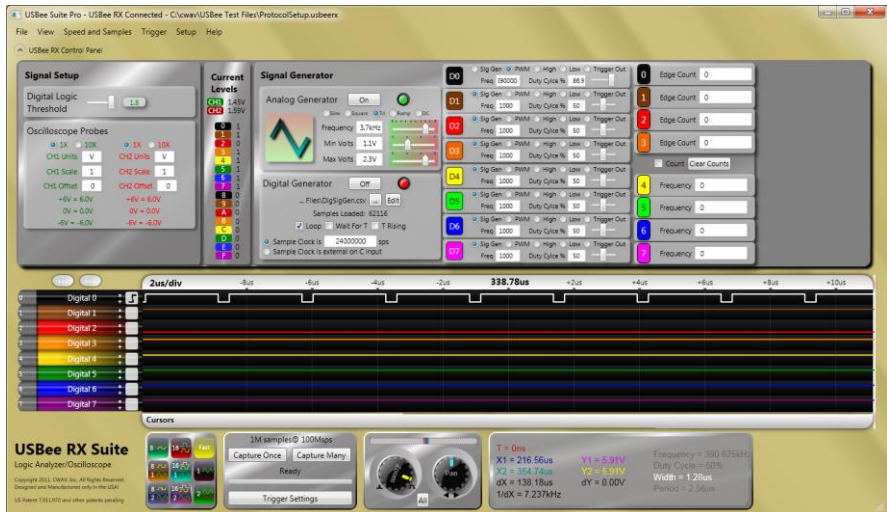
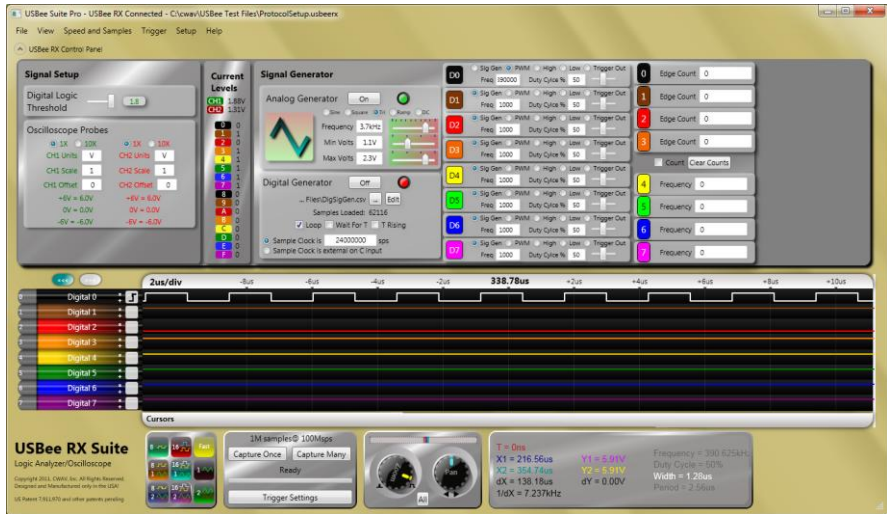
The **Trigger Out** level outputs 0V before the trigger event and 3.3V after the trigger event. This can be used to trigger other equipment to synchronize the captures of data.

## PULSE WIDTH MODULATORS

Each digital output channel can be configured as a Pulse Width Modulator. You specify the frequency in Hz (0 to 390000) desired as well as the duty cycle in percent (0 to 100) of the waveform.

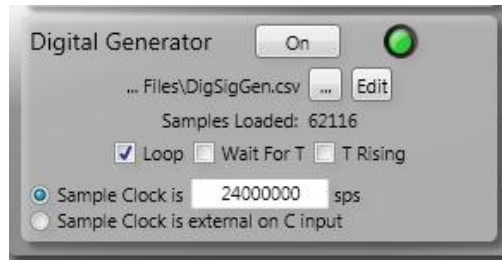
The following shows two examples of a PWM signal with different duty cycles.





## ARBITRARY DIGITAL PATTERN GENERATOR

Each digital output channel can be configured as an arbitrary digital pattern. You specify the pattern by using a Comma Delimited File (CSV) that specifies the Sample Rate (in Hz – 0 to 100000000) and each of the samples. Each pattern can have from 1 to 65535 samples. A CSV file can be created using a simple text editor, or as an output of a program such as Excel.



### SPECIFYING THE PATTERN WITH A CSV FILE

To create a digital pattern, you need to make comma delimited text file. The first item on the first line of the file is the desired sample rate that each of the samples will be clocked out at.

To specify the file to use, select the browse button “...” and choose the file. The file will be parsed and the number of samples loaded and sample rate will be sent to the Digital Signal Generator.

The file format of the CSV files is as follows

```
500000, Sample Rate
23, First sample
51, Second sample
...
84, Last Sample
```

Only the first field of each line is considered as a sample. This allows you to use a spreadsheet that computes the samples in the first column. A sample Excel spreadsheet and the resulting .CSV file is included in the Program Files/CWAV Inc/USBee RX Suite directory for you to use.

For 65535 samples it takes about 30 seconds to download.

Each sample is a decimal value that represents an 8 bit value (from value 0 to 255). These 8 bits are tied to each of the 8 digital outputs: Bit 0 of the sample value goes to the D0 output pin. Bit 7 of the value goes to the D7 output pin and so on. When the pattern is being generated, all 8 digital outputs are changed at the same time to the 8 digital values represented by the sample value.

For example if the sample value is 85 (55 hex), then at that single sample time the outputs will be (from D7 to D0) 0, 1, 0, 1, 0, 1, 0, 1.

Since each of the actual output pins can be independently configured to be Sig Gen, PWM, High, Low or Trigger out, you must make sure the channel is set to Sig Gen in order to see the output of the Signal Generator.

## SAMPLE CLOCKING

The CSV file specifies the desired sample rate of the clocked out digital samples. You can then modify the sample rate by changing the frequency in the sps text box (from 0 to 100000000).

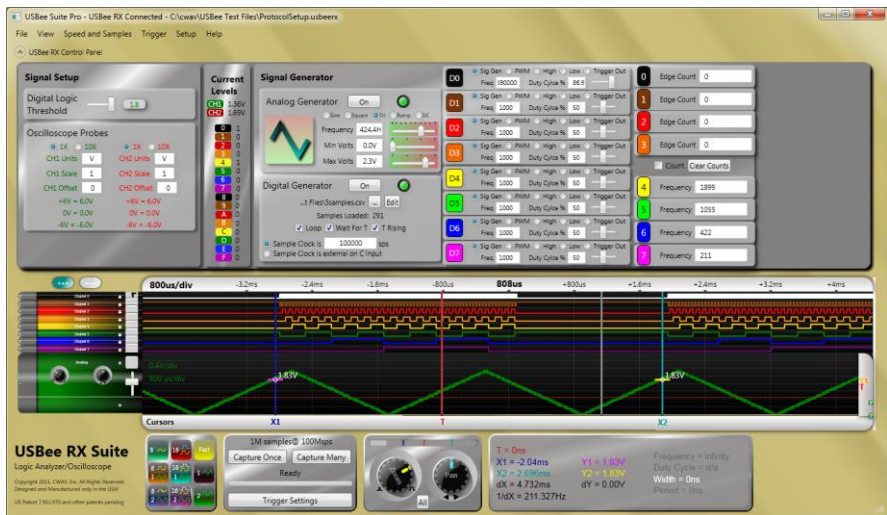
You can also specify to use the external C input as the sample clock by selecting the associated radio button. Every cycle of the C input will clock out the next sample in the sequence.

## WAVEFORM GENERATION CONTROL

The **Wait for T** checkbox (and the associated **T Rising** checkbox to specify the polarity) makes the USBee RX wait for the T input to have an edge before starting the pattern. This allows for synchronizing the [pattern generation with an external event.

The **Loop** checkbox allows you to loop the generated pattern. If the Wait for T checkbox is selected, each cycle waits for the T to be asserted before starting the generation.

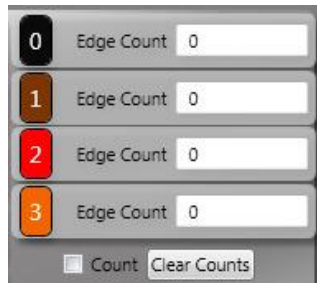
The following screenshot shows the Digital Signal Generator configured to generate 291 samples each time the T signal has a rising edge. In this case the T signal is tied to the CH1 analog signal. The Loop selection makes it repeat, but only after the T signal sees another rising edge. Since the Digital Logic Threshold is set to 1.8V, the T transitions to logic 1 at 1.8V.





## USBEE RX EDGE/PULSE COUNTERS

The USBee RX can independently measure the number of edges that occur on four of the digital input signals (0 through 3). The number of edges counted is shown in the USBee RX Suite control panel as below:



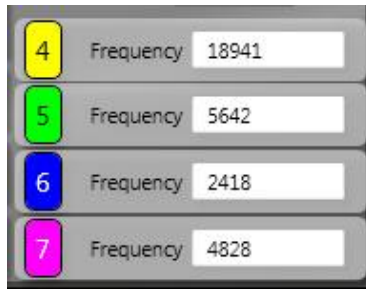
0	Edge Count	0
1	Edge Count	0
2	Edge Count	0
3	Edge Count	0
<input type="checkbox"/> Count		Clear Counts

Counting occurs when the **Count** checkbox is selected. To clear the counts, press the **Clear Counts** button. Remember that the counts are affected by the **Digital Logic Threshold** setting, so you must set this correctly for the signals you are measuring for accurate counts.

Since each Pulse is made up of a rising edge and falling edge pair, you can compute the number of pulses by dividing the Number Of Edges by 2.

## USBEE RX FREQUENCY COUNTERS

The USBee RX can independently measure the frequency four of the digital input signals (3 through 7). The frequency measured is shown in the USBee RX Suite control panel as below:



Remember that the measurements are affected by the **Digital Logic Threshold** setting, so you must set this correctly for the signals you are measuring for accurate counts.

### OVERVIEW

The USBee RX Test Pod System consists of the USBee RX Test Pod connected to a Windows® XP, Vista or 7 PC High Speed USB 2.0 port through the USB cable, and to your circuit using the multicolored test leads and clips. Once connected and installed, the USBee can then be controlled using either the USBee RX Suite Windows Software or your own USBee RX Toolbuilder software.

The USBee RX Test Pod is ideal for students or designers that need to get up and running with High Speed USB immediately. With a mini-B USB connector on one end and signal pin headers on the other, this simple pod will instantly USB 2.0 High-Speed enable your design. Then using the source code libraries, drivers and DLL's that are included you can write your own PC application to control and monitor the signal pins on the pod.

---

### SYSTEM SOFTWARE ARCHITECTURE

The USBee RX Pod is controlled through a set of Windows DLL function calls. These function calls are defined in following sections and provide initialization and data transfer routines. This DLL can be called using a variety of languages, including Visual BASIC, C, C++, etc. We have included a sample application in C and VB that show how you can use the calls to setup and control the pod. You can port this example to any language that can call DLL functions (Delphi, Lab View, ...)

After installing the USBee RX Suite software on your computer, you can then plug in the USBee RX pod. Immediately after plugging in the pod, the operating system finds the INF file installed during installation. This file specifies which driver to load for that device, which is the Windows provided WIBUSB driver. This driver then remains resident in memory until you unplug the device.

Once you run your USBee Toolbuilder application, it will call the functions in the USBEERXSTE.DLL file which is installed in the \Windows\System32 directory for 32-bit Windows, or \Windows\SysWOW directory for 64-bit windows. This DLL will then make the correct calls to the WINUSB driver to perform the USB transfers that are required by the pod.

Both example projects are built using Microsoft Visual Studio 2010 which can be downloaded from the Microsoft web site.

## THE USBEE RX POD HARDWARE



The USBee RX Test Pod has four sets of connectors:

- USB
  - Connects to the PC via 6 foot USB cable
- CH1 and CH2
  - Analog Input Channels
  - BNC connectors for standard oscilloscope probes
  - +/- 60V tolerant
- Outputs connector
  - 11 pins – 1 Ground (GND), 1 Analog Aout, 1 +5V Supply, and 8 Digital Outputs (D0-D7)
  - 0.1" spaced 0.025" square header pins
  - 8 Digital output pins (0-3.3V logic) – D0 – D7
  - +5V output – actually VBus connected directly to the USB VBus signal from the PC through a resettable fuse
  - 1 Aout analog output pin used by the Analog Signal Generator function (0-3V levels)
  - Connects to 11x1 pin test lead set
- Digital Inputs Connector
  - 20 pins (10x2) – 2 Ground (G), Clock Input (C), Trigger Input (T), and 16 Digital inputs (0-F)
  - 0.1" spaced 0.025" square header pins
  - +/-60V tolerant inputs
  - Connects to 10x2 pin test lead set



## INSTALLING THE USBEE RX TOOLBUILDER

The USBee RX Toolbuilder software is included as part of the installation with the USBee RX Suite software and can be downloaded from [www.usbee.com](http://www.usbee.com). Run the setup.exe install program in the downloaded file to install from the web. The install program will install the following USBee Toolbuilder files and drivers into their correct location on your system. Other files will also be installed, but are not necessary for Toolbuilder operation.

---

### USBEE RX TOOLBUILDER PROJECTS

The USBee RX example Toolbuilder projects are located in

`\Program Files\USBee RX Suite\USBee RX Toolbuilder\USBeeRXToolbuilderInC` , and

`\Program Files\USBee RX Suite\USBee RX Toolbuilder\USBeeRXToolbuilderInVB`

The USBee RX Toolbuilder depends on the files in the directories for proper operation, so if you want to relocate the projects to somewhere else, you must maintain the entire file/directory structure.

## USBEE RX TOOLBUILDER FUNCTIONS

This section details the functions that are available in the `usbbeerxste.dll` and defines the parameters to each call. The calling conventions are shown for the C language, but you can find the definitions for Visual Basic in the example file as well.

### INITIALIZING THE USBEE RX POD

#### INITIALIZERX

This routine initializes the USBee RX Pod. This routine must be called before calling any other USBee RX functions.

Calling Convention

```
int InitializeRX( void );
```

Return Value:

0 = Pod Not Found

1 = Pod Initialized

### READING AND SETTING SIGNAL STATES

#### SETSIGNALSRX - SETTING THE USBEE RX OUTPUT SIGNALS

Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API SetSignalsRX( unsigned long mask, unsigned long value );
```

- **mask** is the mask for setting each of the 8 USBee digital Output signals (0 through 7). A signal is not changed if the corresponding bit is a 0. A signal is changed if the corresponding bit is a 1. Channel D0 is bit 0 (lsb) and D7 is bit 7.
- **value** is the digital level driven on the output signals. A signal is driven high (3.3V) if the corresponding bit is a 1. A signal is driven low (0V) if the corresponding bit is a 0. Channel D0 is bit 0 (lsb) and D7 is bit 7.

Return Value:

- 1 = Successful

- 0 = Failure

## GETSIGNALSRX - READING THE USBEE RX INPUT SIGNALS

### Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API GetSignalsRX( unsigned char *digital0to7, unsigned char *digital8toF, unsigned char *analog1, unsigned char *analog2, unsigned char *CandT );
```

- After this call, the variable pointed to by **digital0to7** will hold the digital logic value read on the input signals 0 through 7.
- After this call, the variable pointed to by **digital8toF** will hold the digital logic value read on the input signals 8 through F.
- After this call, the variable pointed to by **analog1** will hold the 8 MSbits of the CH1 ADC. To convert the digital value to voltage,  $V = ((128 - \text{analog1}) * 0.046875)$
- After this call, the variable pointed to by **analog2** will hold the 8 MSbits of the CH2 ADC. To convert the digital value to voltage,  $V = ((128 - \text{analog2}) * 0.046875)$
- After this call, the variable pointed to by **CandT** will hold the digital logic value read on the input signals C and T.

### Return Value:

- 1 = Successful
- 0 = Failure

## GENERATEANALOGWAVEFORMRX – CONTROLLING THE AOUT SIGNAL

### Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API GenerateAnalogWaveformRX( unsigned long SamplesPerSecond, unsigned char SamplesPerCycle, unsigned char *samples );
```

- **SamplesPerSecond** is the sample rate of the output samples and ranges from 1 to 300,000.
- **SamplesPerCycle** is the number of samples that make up a complete cycle in the analog waveform and ranges from 1 to 128 samples.
- **samples** points to a buffer of samples. Each sample is a digital value representing the analog output voltage using the formula  $V_{out} = \text{samples} / 61.429$ .  $V_{out}$  range is from 0 to 3.0V.

### Return Value:

- 1 = Successful

---

## LOGIC ANALYZER AND OSCILLOSCOPE FUNCTIONS

The following API describes the routines that control the Logic Analyzer and Oscilloscope functionality of the USBee RX Test Pod.

---

### SETLOGICTHRESHOLDRX – SETTING THE DIGITAL LOGIC THRESHOLD

This routine sets the digital logic threshold for all of the digital inputs.

Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API SetLogicThresholdRX( float Thresh );
```

where Thresh is the voltage threshold. Digital inputs below this threshold will be interpreted as a logic 0 and voltages above this threshold will be interpreted as a logic 1.

Return Value:

- 1 = Successful
- 0 = Failure

---

### STARTCAPTURERX

This routine starts the pod capturing data at the specified trigger and sample rates.

Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API StartCaptureRX( unsigned long  
bufferSize, unsigned char TriggerPosition, float SampleRate, unsigned long  
channelMask, unsigned char ExternalClockingOn, unsigned char CompressionOn  
);
```

- BufferSize is how many 32-bit samples are taken and ranges from 65536 to 16777216.
- TriggerPosition what percent of the buffer should be prestored before the trigger
- channelMask represent which samples to take:
  - Bit 0: 1 = Sample Digital 0 signal, 0 = ignore this channel
  - Bit 1: 1 = Sample Digital 1 signal, 0 = ignore this channel
  - Etc.
  - Bit 15: 1 = Sample Digital F signal, 0 = ignore this channel
  - Bit 16-23: 255 = Sample Analog channel 1, 0 = ignore analog channel 1
  - Bit 24-31: 255 = Sample Analog channel 2, 0 = ignore analog channel 2
- SampleRate is how many samples per second to capture. Valid rates are in units of samples per second ranging from 10 to 100,000,000.

- ExternalClockOn specifies whether to use the external clock input pin “C” to capture samples (1) , or the internal sample clock as specified above (0).
- CompressionOn specifies whether to use sample compression (1) or not (0).

Return Value:

- 0 = Failed
- 1 = Success

---

## CAPTURESTATUSRX

This routine checks the status of the data capture in progress.

Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API CaptureStatusRX( unsigned char *Full,
unsigned char *Triggered, unsigned char *Running );
```

- Running: 1 = Acquisition is still running, 0 = Acquisition has completed
- Triggered: 1 = Trigger has occurred, 0 = still waiting for the trigger
- Full: The percentage of the buffer that is currently filled. Ranges from 0 to 100.

Return Value:

- 0 = Failed
- 1 = Success

---

## STOPCAPTURERX

This routine terminates a pending capture.

### Calling Convention

```
int StopCaptureRX(void)
```

Return Value:

- 1 = Capture Stopped
- 0 = Stop Failed

---

## TRIGGERNOWRX

This routine immediately triggers a pending capture.

### Calling Convention

```
int TriggerNowRX(void)
```

Return Value:

- 1 = Trigger Occurred
- 0 = Trigger Failed

---

## ENDCAPTURERX

This routine completes the capture and transfers all of the stored data to the PC. This routine should be called once a capture is completed (CaptureStatusRX Running is 0).

### Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API EndCaptureRX( __int64  
*ActualNumberOfSamples, __int64 *TriggerPosition);
```

- ActualNumberOfSamples specifies how many samples were captured at the given sample rate
- TriggerPosition specifies which sample in the captured buffer contains the trigger event referenced from the beginning of the samples.

Return Value:

- 1 = End successful

- 0 = Trigger Failed

## SAMPLEDATA

This routine returns the 4 byte value of a particular sample. The low 2 bytes contain the 16 digital channels. The high two bytes contain two 8-bit ADC values for the two analog channels.

Calling Convention

```
unsigned long SampleData( __int64 index )
```

Index: sample number to return

Return Value:

Value of the given sample

## FINDNEXTEDGE

This routine scans through the sample buffer and finds the next edge on a given channel. This routine should be used in the cases of compressed data instead of individual SampleData calls.

Calling Convention

```
CWAV_IMPORT __int64 CWAV_API FindNextEdge( __int64 UCSample, unsigned long
channelmask, unsigned long direction );
```

UCSample is the sample number at which to start looking for an edge.

Channelmask is which channels to look for an edge on. A 1 in a bit position specifies to use that channel. Bit 0 = digital channel 0, Bit 15 = digital channel F.

Direction specifies forward (1) in time, or backward (0) in time.

Return Value:

Sample number where the edge was found.

## SETTRIGGERSRX

This routine sets the trigger condition to use during the next capture.

Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API SetTriggersRX( int TrigXEnabled, int
TrigYEnabled, int TrigXorYEnabled, int TrigXandYEnabled, int
TrigXthenYEnabled, int TrigYthenXEnabled,
```

```

        int TrigX_DigitalEdgeEnabled, int
TrigX_AnalogEdgeEnabled, int TrigX_QualifyDigitalEnabled, int
TrigX_QualifyAnalogEnabled, int TrigX_QualifyTimeEnabled,
        int TrigX_InvertDigitalQualifierEnabled, int
TrigX_InvertAnalogQualifierEnabled, int TrigX_InvertTimeQualifierEnabled,
int TrigX_DigitalEdgeChannel,
        int TrigX_DigitalEdgeRising, int
TrigX_AnalogEdgeChannel, int TrigX_AnalogEdgeRising, int
TrigX_AnalogQualifierChannel, int TrigX_AnalogQualifierFrom,
        int TrigX_AnalogQualifierTo, int
TrigX_AnalogTriggerLevel, long TrigX_DigitalQualifierChannelMask, long
TrigX_DigitalQualifierFrom, long TrigX_DigitalQualifierTo,

        int TrigY_DigitalEdgeEnabled, int
TrigY_AnalogEdgeEnabled, int TrigY_QualifyDigitalEnabled, int
TrigY_QualifyAnalogEnabled, int TrigY_QualifyTimeEnabled,
        int TrigY_InvertDigitalQualifierEnabled, int
TrigY_InvertAnalogQualifierEnabled, int TrigY_InvertTimeQualifierEnabled,
int TrigY_DigitalEdgeChannel,
        int TrigY_DigitalEdgeRising, int
TrigY_AnalogEdgeChannel, int TrigY_AnalogEdgeRising, int
TrigY_AnalogQualifierChannel, int TrigY_AnalogQualifierFrom,
        int TrigY_AnalogQualifierTo, int
TrigY_AnalogTriggerLevel, long TrigY_DigitalQualifierChannelMask, long
TrigY_DigitalQualifierFrom, long TrigY_DigitalQualifierTo

    );

```

- TriggerYEnabled: 1 = Trigger when Trig Y is present
- TriggerXEnabled: 1 = Trigger when Trig X is present
- TrigXorYEnabled: 1 = Trigger when Trig X or Y is present
- TrigXandYEnabled: 1 = Trigger when Trig X and Y is present
- TrigXthenYEnabled: 1 = Trigger when Trig X followed by Y is present
- TrigYthenXEnabled: 1 = Trigger when Trig Y followed by X is present

Return Value:

- 1 = Trigger Set successful
- 0 = Trigger Set Failed

## DECODEUSB

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

Calling Convention

```

CWBV_IMPORT int CWBV_API DecodeUSB (unsigned long *reserved1, unsigned char
*OutFilename, unsigned char *InlineFilename, __int64 StartSample, __int64
EndSample, long ShowEndpoint, long ShowAddress, long DPlus, long DMinus,
long Speed, long Rate, long SOF, long delimiter, long showall, long hex,

```



```
char *ProtocolDefinitionFilename, char *ProtocolOutputFilename, char
*ErrorString);
```

- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- ShowEndpoint: 999 = show all traffic, otherwise show only this USB endpoint number traffic
- ShowAddress: 999 = show all USB devices, otherwise only show the USB device with this USB address
- DPlus: Which signal (0 – 15) to use for the D Plus signal
- DMinus: Which signal (0 – 15) to use for the D Minus signal
- Speed: 0 = Low Speed USB, 1 = Full Speed USB
- Rate is as follows:
  - 69 = 600 Msps
  - 39 = 300 Msps
  - 29 = 200 Msps
  - 108 = 100 Msps
  - 58 = 50 Msps
  - 28 = 20 Msps
  - 107 = 10 Msps
  - 57 = 5 Msps
  - 27 = 2Msps
  - 106 = 1 Msps
  - 56 = 500ksps
  - 26 = 200ksps
  - 105 = 100ksps
- SOF: 0 = do not show the SOF (Start of Frames), 1 = show SOFs
- Delimiter: 0 = no delimiter, 1 = Comma delimiter, 2 = Space delimiter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

## DECODESPI

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

### Calling Convention

```
CWAV_IMPORT int CWAV_API DecodeSPIVariable (unsigned long *reserved1,
unsigned char *OutFilename,unsigned char *InlineMOSIFilename,unsigned char
*InlineMISOFilename, __int64 StartSample, __int64 EndSample, long Rate,
unsigned long SS,unsigned long SCK,unsigned long tMOSI,unsigned long tMISO,
unsigned long MISOEdge,unsigned long MOSIEdge, unsigned long
delimiter,unsigned long hex,unsigned long UseSS, unsigned long SSLevel,
long BytesPerLine, long BitsPerByte, char *ProtocolDefinitionFilename, char
*ProtocolOutputFilename, char *ErrorString);
```

- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- Rate is as follows:
  - 69 = 600 Msps
  - 39 = 300 Msps
  - 29 = 200 Msps
  - 108 = 100 Msps
  - 58 = 50 Msps
  - 28 = 20 Msps
  - 107 = 10 Msps
  - 57 = 5 Msps
  - 27 = 2Msps
  - 106 = 1 Msps
  - 56 = 500kps
  - 26 = 200kps
  - 105 = 100kps
- SS: Which signal (0 – 15) to use for the Slave Select signal
- SCK: Which signal (0 – 15) to use for the clock signal
- MISO: Which signal (0 – 15) to use for the MISO signal
- MOSI: Which signal (0 – 15) to use for the MOSI signal
- MOSIEdge: 0 = use falling edge of SCK to sample data on MOSI, 1 = use rising edge
- MISOEdge: 0 = use falling edge of SCK to sample data on MISO, 1 = use rising edge
- Delimiter: 0 = no delimiter, 1 = Comma delimiter, 2 = Space delimiter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex
- UseSS: 0 = don't use an SS signal, 1 = use the SS signal
- BytesPerLine: How many output words are on each output line.
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

---

## DECODEI2C

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

Calling Convention

```
CWAV_IMPORT int CWAV_API DecodeI2C (unsigned long *reserved1, unsigned char
*OutFilename, unsigned char *InlineSDAFilename, __int64 StartSample,
__int64 EndSample, long Rate, unsigned long SDA, unsigned long SCL, long
showack, long delimiter, long showall, long hex, char
*ProtocolDefinitionFilename, char *ProtocolOutputFilename, char
*ErrorString);
```

- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- Rate is as follows:
  - 69 = 600 Msps
  - 39 = 300 Msps
  - 29 = 200 Msps
  - 108 = 100 Msps
  - 58 = 50 Msps
  - 28 = 20 Msps
  - 107 = 10 Msps
  - 57 = 5 Msps
  - 27 = 2Msps
  - 106 = 1 Msps
  - 56 = 500kps
  - 26 = 200kps
  - 105 = 100kps
- SDA: Which signal (0 – 15) to use for the SDA signal
- SCL: Which signal (0 – 15) to use for the SCL signal
- ShowAck: 0 = Do not show each byte ACK values, 1 = show the ACK value after each byte
- Delimiter: 0 = no delimiter, 1 = Comma delimiter, 2 = Space delimiter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

## DECODECAN

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

### Calling Convention

```
CWAV_IMPORT int CWAV_API DecodeCAN (unsigned long *reserved1, unsigned char
*OutFilename, unsigned char *OutFilename, __int64 StartSample, __int64
EndSample, unsigned long Rate, unsigned long Channel, unsigned long
BitRate, unsigned long maxID, unsigned long minID, long delimiter, long
showall, long Phex, char *ProtocolDefinitionFilename, char
*ProtocolOutputFilename, char *ErrorString);
```

- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- Rate is as follows:
  - 69 = 600 Msps
  - 39 = 300 Msps
  - 29 = 200 Msps
  - 108 = 100 Msps
  - 58 = 50 Msps
  - 28 = 20 Msps
  - 107 = 10 Msps
  - 57 = 5 Msps
  - 27 = 2Msps
  - 106 = 1 Msps
  - 56 = 500kps
  - 26 = 200kps
  - 105 = 100kps
- Channel: Which signal (0 – 15) to use for the CAN signal
- BitRate: The value of the bit rate in bits per second (for 250kbps use 250000)
- MaxID: 0 = show all packets, otherwise this is the maximum ID to display
- MinID: 0 = show all packets, otherwise this is the minimum ID to display
- Delimiter: 0 = no delimiter, 1 = Comma delimiter, 2 = Space delimiter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- PHex: 0 = display data in decimal, 1 = display data in hex
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

---

## DECODE1WIRE

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

Calling Convention

```
CWAV_IMPORT int CWAV_API Decode1Wire (unsigned long *reserved1, unsigned
char *OutFilename, unsigned char *InlineFilename, _int64 StartSample,
_int64 EndSample, long Rate, unsigned long Signal, long delimiter, long
showall, long hex, char *ProtocolDefinitionFilename, char
*ProtocolOutputFilename, char *ErrorString);
```

- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- Rate is as follows:
  - 69 = 600 Msps
  - 39 = 300 Msps
  - 29 = 200 Msps
  - 108 = 100 Msps
  - 58 = 50 Msps
  - 28 = 20 Msps
  - 107 = 10 Msps
  - 57 = 5 Msps
  - 27 = 2Msps
  - 106 = 1 Msps
  - 56 = 500ksps
  - 26 = 200ksps
  - 105 = 100ksps
- Signal: Which signal (0 – 15) to use for the 1-Wire signal
- Delimiter: 0 = no delimiter, 1 = Comma delimiter, 2 = Space delimiter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

## DECODEPARALLEL

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

### Calling Convention

```
CWAV_IMPORT int CWAV_API DecodeParallel (unsigned long *reserved1, unsigned
char *OutFilename, unsigned char *InlineFilename, __int64 StartSample,
__int64 EndSample, long Rate, unsigned long Channels, unsigned long Clock,
unsigned long UseCLK, long CLKEdge, unsigned long delimiter, unsigned long
hex, long BytesPerLine, char *ProtocolDefinitionFilename, char
*ProtocolOutputFilename, char *ErrorString);
```

- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- Rate is as follows:
  - 69 = 600 Msps
  - 39 = 300 Msps
  - 29 = 200 Msps
  - 108 = 100 Msps
  - 58 = 50 Msps
  - 28 = 20 Msps
  - 107 = 10 Msps
  - 57 = 5 Msps
  - 27 = 2 Msps
  - 106 = 1 Msps
  - 56 = 500ksps
  - 26 = 200ksps
  - 105 = 100ksps
- Channels: Bit mask which represents which signals are part of the parallel data bus. Bit 0 is Pod signal 0. Bit 15 is pod signal F.
- Clock: Which signal (0 – 15) to use for the clock signal
- UseCLK: 0 – don't use the Clock signal above, 1 – use the Clock signal above to qualify the samples
- CLKEdge: 0 = use falling edge of the Clock to sample data, 1 = use rising edge
- Delimiter: 0 = no delimiter, 1 = Comma delimiter, 2 = Space delimiter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex
- BytesPerLine: How many output words are on each output line.
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.

- `ErrorString` – string that holds an error description of the routine returns an error.

Return Value:

- `TRUE` – No Error during processing
- `FALSE` – Error while processing. The `ErrorString` contains a description of the error to present to the user.

## DECODESERIAL

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the `StartCapture` routine.

Calling Convention

```
CWAV_IMPORT int CWAV_API DecodeSerial (unsigned long *reserved1, unsigned
char *OutFilename, unsigned char *InlineFilename, __int64 StartSample,
__int64 EndSample, unsigned long Rate, unsigned long Channel, unsigned long
AlignValue, unsigned long AlignEdge, unsigned long AlignChannel, unsigned
long UseAlignChannel, unsigned long ClockChannel, unsigned long ClockEdge,
unsigned long BitsPerValue, unsigned long MSBFirst, unsigned long
delimiter, unsigned long hex, long BytesPerLine, char
*ProtocolDefinitionFilename, char *ProtocolOutputFilename, char
*ErrorString);
```

- `OutFilename`: pointer to the filename string to write the decoded data to.
- `StartSample`: the index of the first sample to start decoding
- `EndSample`: the index of the last sample to decode
- `Rate` is the rate at which samples were taken during `StartCapture`:
  - 69 = 600 Msps
  - 39 = 300 Msps
  - 29 = 200 Msps
  - 108 = 100 Msps
  - 58 = 50 Msps
  - 28 = 20 Msps
  - 107 = 10 Msps
  - 57 = 5 Msps
  - 27 = 2 Msps
  - 106 = 1 Msps
  - 56 = 500kps
  - 26 = 200kps
  - 105 = 100kps
- `Channel`: Which signal (0 – 15) to use for the serial signal
- `AlignValue`: When using word aligning, bus value which is used for aligning the serial stream to byte boundaries.
- `AlignEdge`: When using an external signal for aligning, 0 = falling edge, 1 = rising edge.
- `AlignChannel`: When using an external signal for aligning, which signal (0 – 15) to use for the align signal

- UseAlignChannel: 0 = use word aligning, 1 = use external align signal
- ClockChannel: Which signal (0 – 15) to use for the clock signal
- CLKEdge: 0 = use falling edge of the Clock to sample data, 1 = use rising edge
- BitsPerValue: how many bits are in each word of the serial stream
- MSBFirst: 0 = LSBit is sent first, 1 = MSBit is sent first
- Delimeter: 0 = no delimiter, 1 = Comma delimiter, 2 = Space delimiter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex
- BytesPerLine: How many output words are on each output line.
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

## DECODEASYNC

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

Calling Convention

```
CWAV_IMPORT int CWAV_API DecodeASYNC (unsigned long *reserved1, unsigned
char *OutFilename, unsigned char *OutTxFilename, unsigned char
*OutRxFilename, __int64 StartSample, __int64 EndSample, long Rate, unsigned
long TxChannel, unsigned long RxChannel, unsigned long BaudRate, unsigned
long Parity, unsigned long DataBits, unsigned long Invert, unsigned long
delimiter, unsigned long hex, unsigned long ascii, long BytesPerLine, char
*ProtocolDefinitionFilename, char *ProtocolOutputFilename, char
*ErrorString);
```

- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- Rate is the rate at which samples were taken during StartCapture:
  - 69 = 600 Msps
  - 39 = 300 Msps
  - 29 = 200 Msps
  - 108 = 100 Msps
  - 58 = 50 Msps
  - 28 = 20 Msps
  - 107 = 10 Msps
  - 57 = 5 Msps



- 27 = 2Msps
- 106 = 1 Msps
- 56 = 500ksps
- 26 = 200ksps
- 105 = 100ksps
- TX and RX Channels: Bit mask which represents which signal to decode. Bit 0 is Pod signal 0. Bit 15 is pod signal F.
- BaudRate: Baud Rate in bits per second (19.2K = 19200)
- Parity: 0 = No parity, 1 = Mark, 2 = Space, 3 = Even, 4 = Odd, 5 = Ignore
- DataBits: Number of data bits (4 to 24)
- Delimeter: 0 = no delimiter, 1 = Comma delimiter, 2 = Space delimiter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Invert: 1 = invert the logic of the signal before processing
- Hex: 0 = display data in decimal, 1 = display data in hex
- ASCII: 0 = show byte values, 1 = show ASCII equivalent
- BytesPerLine: How many output words are on each output line.
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

## DECODESETNAME

This routine sets the string that is output during any of the above decoders and can represent a unique identifier for that bus.

Calling Convention

```
int DecodeSetName (char *name);
```

## DIGITAL PATTERN GENERATOR FUNCTIONS

The following API describes the routines that control the digital pattern generator functionality of the USBee RX Test Pod.

---

## GENERATEPWMWAVEFORMRX

This routine sets the digital output channel specified to a PWM (pulse width modulator) signal.

### Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API GeneratePWMWaveformRX( unsigned char  
channel, unsigned long samplespersecond, unsigned char dutycycle );
```

- Channel: which channel (0 to 7) to setup as a PWM output
- Samplespersecond: how many samples per second for the PWM signal
- Dutycycle: a value from 0 to 100 for the low to high percentage of the PWM signal.

### Return Value:

- 1 = Successful
- 0 = Failure

---

## LOADDIGITALWAVEFORMRX

This routine sets the digital output channel specified to a PWM (pulse width modulator) signal.

### Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API LoadDigitalWaveformRX( unsigned long  
numberofsamples, unsigned char *samples );
```

- Numberofsamples: a value from 1 to 61000 indicating the number of samples in the pattern
- Samples: a pointer to the buffer of samples. Each sample is a byte and are clocked out on consecutive sample clocks. Bit 0 of each byte is output channel 0, bit 7 is output channel 7.

### Return Value:

- 1 = Successful
- 0 = Failure

---

## GENERATEDIGITALWAVEFORMRX

This routine sets the digital output channel specified to a PWM (pulse width modulator) signal.

### Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API GenerateDigitalWaveformRX( unsigned char
channelmask, unsigned char GenerateOn, unsigned char loop, unsigned char
waitT, unsigned char Trising, unsigned char externalclock, unsigned char
Crising, unsigned long samplespersecond);
```

- Channelmask: which channels to use for the output of the digital pattern generator. Bit 0 is channel 0 and Bit 7 is channel 7. 1 = output digital pattern on this channel, 0 = no output.
- GenerateOn = 1 to turn on the pattern generator, 0 turns it off.
- Loop: 1 = loop the pattern at the end. 0 = No loop
- waitT: Waits to start the pattern until the T signal shows an edge
- Trising: indicates the T signal must be a rising edge (1) or falling edge (0) to trigger the pattern
- External Clock: 1 indicates to use the digital input C to clock out the pattern instead of the internal sample clock
- Crising: 1 indicates the external clock C rising edge, 0 = falling edge
- Samplespersecond is the number of samples per second to clock out the digital pattern (for internal clocking mode).

Return Value:

- 1 = Successful
- 0 = Failure

---

## FREQUENCY AND PULSE COUNTER FUNCTIONS

The following API describes the routines that control the frequency and pulse counter functionality of the USBee RX Test Pod.

---

### CLEARCOUNTSRX

This routine clears the frequency and pulse counts.

Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API ClearCountsRX( void );
```

Return Value:

- 1 = Successful
- 0 = Failure

---

### ENABLECOUNTSRX

This routine enables the frequency and pulse counting.

## Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API EnableCountsRX( void );
```

Return Value:

- 1 = Successful
- 0 = Failure

---

## DISABLECOUNTSRX

This routine disables the frequency and pulse counting.

## Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API DisableCountsRX( void );
```

Return Value:

- 1 = Successful
- 0 = Failure

---

## GETFREQUENCYANDCOUNTSRX

This routine returns the frequency and pulse counting values.

## Calling Convention

```
CWAV_IMPORT unsigned long CWAV_API GetFrequencyAndCountsRX( unsigned long  
*edgecounts0, unsigned long *edgecounts1, unsigned long *edgecounts2,  
unsigned long *edgecounts3, unsigned long *freq4, unsigned long *freq5,  
unsigned long *freq6, unsigned long *freq7 );
```

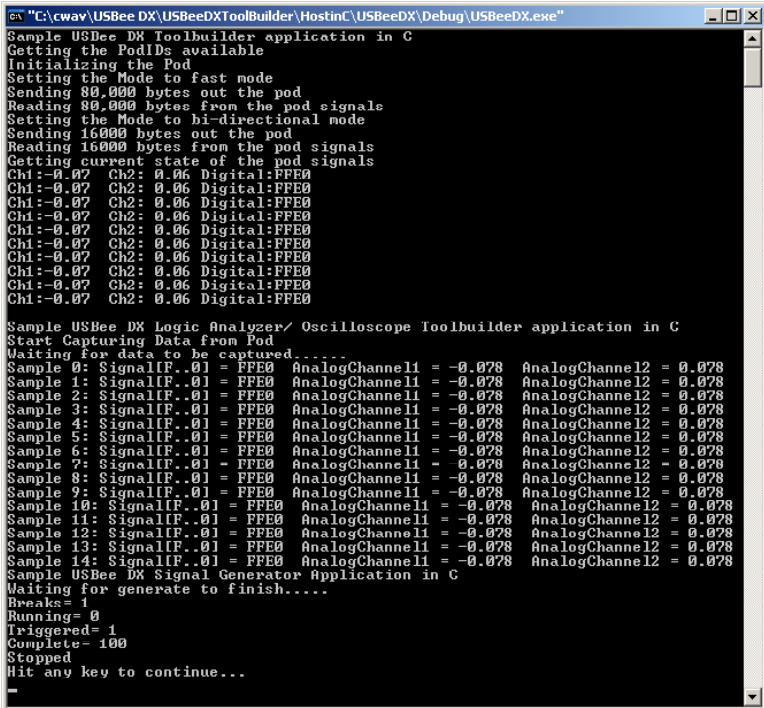
- Edgecounts0: how many edges were counted on digital channel 0
- Edgecounts1: how many edges were counted on digital channel 1
- Edgecounts2: how many edges were counted on digital channel 2
- Edgecounts3: how many edges were counted on digital channel 3
- Freq4: the frequency measured on digital channel 4
- Freq5: the frequency measured on digital channel 5
- Freq6: the frequency measured on digital channel 6
- Freq7: the frequency measured on digital channel 7

Return Value:

- 1 = Successful
- 0 = Failure

## EXAMPLE C CODE

The following code listing is an example in very simple C that calls the DLL functions. It is a Command Prompt program that generates the following output when run.



```
C:\cwav\USBee DX\USBeeDXToolBuilder\HostinC\USBeeDX\Debug\USBeeDX.exe"
Sample USDee DX Toolbuilder application in C
Getting the PodIDs available
Initializing the Pod
Setting the Mode to fast mode
Sending 80,000 bytes out the pod
Reading 80,000 bytes from the pod signals
Setting the Mode to bi-directional mode
Sending 16000 bytes out the pod
Reading 16000 bytes from the pod signals
Getting current state of the pod signals
Ch1:-0.07 Ch2: 0.06 Digital:FFE0
Ch1:-0.07 Ch2: 0.06 Digital:FFE0
Ch1:-0.07 Ch2: 0.06 Digital:FFE0
Ch1:-0.07 Ch2: 0.06 Digital:FFE0
Ch1:-0.07 Ch2: 0.06 Digital:FFE0
Ch1:-0.07 Ch2: 0.06 Digital:FFE0
Ch1:-0.07 Ch2: 0.06 Digital:FFE0
Ch1:-0.07 Ch2: 0.06 Digital:FFE0
Ch1:-0.07 Ch2: 0.06 Digital:FFE0
Ch1:-0.07 Ch2: 0.06 Digital:FFE0
Sample USDee DX Logic Analyzer/ Oscilloscope Toolbuilder application in C
Start Capturing Data from Pod
Waiting for data to be captured.....
Sample 0: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 1: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 2: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 3: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 4: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 5: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 6: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 7: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 8: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 9: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 10: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 11: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 12: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 13: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample 14: SignalIF..01 = FFE0 AnalogChannel1 = -0.078 AnalogChannel2 = 0.078
Sample USDee DX Signal Generator Application in C
Waiting for generate to finish.....
Breaks= 1
Running= 0
Triggered= 1
Complete= 100
Stopped
Hit any key to continue...
=
```

```

// USBeeRXToolbuilderExampleC.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "conio.h"
#include "windows.h"

#define CWA_V_API __stdcall
#define CWA_V_EXPORT __declspec(dllexport)
#define CWA_V_IMPORT __declspec(dllimport)

CWA_V_IMPORT unsigned long CWA_V_API InitializeRX( void );
CWA_V_IMPORT unsigned long CWA_V_API GetSignalsRX( unsigned char *digital0to7, unsigned char
    *digital8toF, unsigned char *analog1, unsigned char *analog2, unsigned char *CandT );
CWA_V_IMPORT unsigned long CWA_V_API SetSignalsRX( unsigned long mask, unsigned long value );
CWA_V_IMPORT unsigned long CWA_V_API GenerateAnalogWaveformRX( unsigned long SamplesPerSecond,
    unsigned char SamplesPerCycle, unsigned char *samples );
CWA_V_IMPORT unsigned long CWA_V_API GeneratePWMWaveformRX( unsigned char channel, unsigned long
    samplespersecond, unsigned char dutycycle );
CWA_V_IMPORT unsigned long CWA_V_API LoadDigitalWaveformRX( unsigned long numberofsamples, unsigned
    char *samples );
CWA_V_IMPORT unsigned long CWA_V_API GenerateDigitalWaveformRX( unsigned char channelmask, unsigned
    char GenerateOn, unsigned char loop, unsigned char waitT, unsigned char Trising,
    unsigned char externalclock, unsigned char Crising, unsigned long samplespersecond);
CWA_V_IMPORT unsigned long CWA_V_API GetFrequencyAndCountsRX( unsigned long *edgecounts0, unsigned
    long *edgecounts1, unsigned long *edgecounts2, unsigned long *edgecounts3, unsigned long
    *freq4, unsigned long *freq5, unsigned long *freq6, unsigned long *freq7 );
CWA_V_IMPORT unsigned long CWA_V_API ClearCountsRX( void );
CWA_V_IMPORT unsigned long CWA_V_API EnableCountsRX( void );
CWA_V_IMPORT unsigned long CWA_V_API DisableCountsRX( void );
CWA_V_IMPORT unsigned long CWA_V_API SetLogicThresholdRX( float Thresh );
CWA_V_IMPORT unsigned long CWA_V_API StartCaptureRX( unsigned long buffersize, unsigned char
    TriggerPosition, float SampleRate, unsigned long channelmask, unsigned char
    ExternalClockingOn, unsigned char CompressionOn );
CWA_V_IMPORT unsigned long CWA_V_API CaptureStatusRX( unsigned char *Full, unsigned char *Triggered,
    unsigned char *Running );
CWA_V_IMPORT unsigned long CWA_V_API StopCaptureRX( void );
CWA_V_IMPORT unsigned long CWA_V_API TriggerNowRX( void );
CWA_V_IMPORT unsigned long CWA_V_API EndCaptureRX( __int64 *ActualNumberOfSamples, __int64
    *TriggerPosition);
CWA_V_IMPORT unsigned long CWA_V_API SampleData( __int64 index );
CWA_V_IMPORT __int64 CWA_V_API FindNextEdge( __int64 UCSample, unsigned long channelmask, unsigned
    long direction );
CWA_V_IMPORT unsigned long CWA_V_API SetTriggersRX( int TrigXEnabled, int TrigYEnabled, int
    TrigXorYEnabled, int TrigXandYEnabled, int TrigXthenYEnabled, int TrigYthenXEnabled,

    int TrigX_DigitalEdgeEnabled, int TrigX_AnalogEdgeEnabled, int
    TrigX_QualifyDigitalEnabled, int TrigX_QualifyAnalogEnabled, int
    TrigX_QualifyTimeEnabled,
    int TrigX_InvertDigitalQualifierEnabled, int TrigX_InvertAnalogQualifierEnabled, int
    TrigX_InvertTimeQualifierEnabled, int TrigX_DigitalEdgeChannel,
    int TrigX_DigitalEdgeRising, int TrigX_AnalogEdgeChannel, int TrigX_AnalogEdgeRising,
    int TrigX_AnalogQualifierChannel, int TrigX_AnalogQualifierFrom,
    int TrigX_AnalogQualifierTo, int TrigX_AnalogTriggerLevel, long
    TrigX_DigitalQualifierChannelMask, long TrigX_DigitalQualifierFrom, long
    TrigX_DigitalQualifierTo,

    int TrigY_DigitalEdgeEnabled, int TrigY_AnalogEdgeEnabled, int
    TrigY_QualifyDigitalEnabled, int TrigY_QualifyAnalogEnabled, int
    TrigY_QualifyTimeEnabled,
    int TrigY_InvertDigitalQualifierEnabled, int TrigY_InvertAnalogQualifierEnabled, int
    TrigY_InvertTimeQualifierEnabled, int TrigY_DigitalEdgeChannel,
    int TrigY_DigitalEdgeRising, int TrigY_AnalogEdgeChannel, int TrigY_AnalogEdgeRising,
    int TrigY_AnalogQualifierChannel, int TrigY_AnalogQualifierFrom,
    int TrigY_AnalogQualifierTo, int TrigY_AnalogTriggerLevel, long
    TrigY_DigitalQualifierChannelMask, long TrigY_DigitalQualifierFrom, long
    TrigY_DigitalQualifierTo

);

// Protocol Decoders
CWA_V_IMPORT int CWA_V_API DecodeSerial( unsigned long *reserved1, unsigned char *OutFilename,
    unsigned char *InlineFilename, __int64 StartSample, __int64 EndSample, unsigned long
    Rate, unsigned long Channel, unsigned long AlignValue, unsigned long AlignEdge, unsigned

```

```

    long AlignChannel,unsigned long UseAlignChannel, unsigned long ClockChannel,unsigned long
    ClockEdge, unsigned long BitsPerValue, unsigned long MSBFirst, unsigned long
    delimiter,unsigned long hex, long BytesPerLine, char *ProtocolDefinitionFilename, char
    *ProtocolOutputFilename, char *ErrorString);

CWA_V_IMPORT int CWA_V_API Decode1Wire (unsigned long *reserved1, unsigned char *OutFilename, unsigned
char *InlineFilename, __int64 StartSample, __int64 EndSample, long Rate, unsigned long
Signal, long delimiter, long showall, long hex, char *ProtocolDefinitionFilename, char
*ProtocolOutputFilename, char *ErrorString);

CWA_V_IMPORT int CWA_V_API DecodeI2S (unsigned long *reserved1, unsigned char *OutFilename, unsigned
char *InlineFilename, __int64 StartSample, __int64 EndSample, unsigned long Rate,
unsigned long Channel, long BitOffset, unsigned long AlignValue, unsigned long AlignEdge,
unsigned long AlignChannel,unsigned long UseAlignChannel, unsigned long
ClockChannel,unsigned long ClockEdge, unsigned long BitsPerValue, unsigned long MSBFirst,
unsigned long delimiter,unsigned long hex, long BytesPerLine, char
*ProtocolDefinitionFilename, char *ProtocolOutputFilename, char *ErrorString);

CWA_V_IMPORT int CWA_V_API DecodeASYN (unsigned long *reserved1, unsigned char *OutFilename,unsigned
char *OutTxFilename,unsigned char *OutRxFilename, __int64 StartSample, __int64 EndSample,
long Rate, unsigned long TxChannel, unsigned long RxChannel, unsigned long BaudRate,
unsigned long Parity, unsigned long DataBits, unsigned long Invert, unsigned long
delimiter,unsigned long hex,unsigned long ascii, long BytesPerLine, char
*ProtocolDefinitionFilename, char *ProtocolOutputFilename, char *ErrorString);

CWA_V_IMPORT int CWA_V_API DecodePS2 (unsigned long *reserved1, unsigned char *OutFilename, unsigned
char *HostFilename, unsigned char *DeviceFilename, __int64 StartSample, __int64
EndSample, long Rate, unsigned long DataChannel, unsigned long ClockChannel, unsigned
long MSBFirst, long hex, char *ProtocolDefinitionFilename, char *ProtocolOutputFilename,
char *ErrorString);

CWA_V_IMPORT int CWA_V_API DecodeUSB (unsigned long *reserved1, unsigned char *OutFilename,unsigned
char *InlineFilename, __int64 StartSample, __int64 EndSample, long ShowEndpoint, long
ShowAddress, long DPlus, long DMinus, long Speed, long Rate, long SOF, long delimiter,
long showall, long hex, char *ProtocolDefinitionFilename, char *ProtocolOutputFilename,
char *ErrorString);

CWA_V_IMPORT int CWA_V_API DecodeSPIVariable (unsigned long *reserved1, unsigned char
*OutFilename,unsigned char *InlineMOSIFilename,unsigned char *InlineMISOFilename, __int64
StartSample, __int64 EndSample, long Rate, unsigned long SS,unsigned long SCK,unsigned
long tMOSI,unsigned long tMISO, unsigned long MISOEdge,unsigned long MOSIEdge, unsigned
long delimiter,unsigned long hex,unsigned long UseSS, unsigned long SSLevel, long
BytesPerLine, long BitsPerByte, char *ProtocolDefinitionFilename, char
*ProtocolOutputFilename, char *ErrorString);

CWA_V_IMPORT int CWA_V_API DecodeI2C (unsigned long *reserved1, unsigned char *OutFilename, unsigned
char *InlineSDAFilename, __int64 StartSample, __int64 EndSample, long Rate, unsigned long
SDA, unsigned long SCL, long showack, long delimiter, long showall, long hex, char
*ProtocolDefinitionFilename, char *ProtocolOutputFilename, char *ErrorString);

CWA_V_IMPORT int CWA_V_API DecodeCAN (unsigned long *InputDecodeBuffer, unsigned char *OutFilename,
unsigned char *InlineFilename, __int64 StartSample, __int64 EndSample, unsigned long
Rate, unsigned long Channel, unsigned long BitRate, unsigned long maxID, unsigned long
minID, long delimiter, long showall, long Phex, char *ProtocolDefinitionFilename, char
*ProtocolOutputFilename, char *ErrorString);

CWA_V_IMPORT int CWA_V_API DecodeParallel (unsigned long *reserved1, unsigned char *OutFilename,
unsigned char *InlineFilename, __int64 StartSample, __int64 EndSample, long Rate,
unsigned long Channels,unsigned long Clock, unsigned long UseCLK, long CLKEdge, unsigned
long delimiter,unsigned long hex, long BytesPerLine, char *ProtocolDefinitionFilename,
char *ProtocolOutputFilename, char *ErrorString);

CWA_V_IMPORT int CWA_V_API DecodeSetName (char *name);

float AnalogCountsToVolts( unsigned char counts )
{
    return ((128 - counts) * 0.046875);
}

int _tmain(int argc, _TCHAR* argv[])
{
    printf( "USBe RX Toolbuilder Example in C\n");

    printf( "Initializing USBe RX.\n");

    if (!InitializerRX())

```

```

{
    printf( "No USBee RX Found.\n");
}
else
{
    printf( "USBee RX Found and Initialized.\n");
    unsigned char D0to7;
    unsigned char D8toF;
    unsigned char CH1;
    unsigned char CH2;
    unsigned char CandT;

    printf( "Setting Digital Outputs.\n");
    SetSignalsRX( 255, 0 );
    SetSignalsRX( 255, 0xFF );
    SetSignalsRX( 255, 0xAA );
    SetSignalsRX( 255, 0x55 );

    printf( "Reading Input Channels.\n");
    GetSignalsRX( &D0to7, &D8toF, &CH1, &CH2, &CandT );
    printf("Digital Channels:%02X%02X CH1:%f CH2:%f T:%d C:%d\n", D8toF,
D0to7, AnalogCountsToVolts( CH1 ), AnalogCountsToVolts( CH2 ), CandT & 1, (CandT >> 1) &
1);

    printf( "Generating Analog Waveform.\n");
    unsigned char samplebuffer[128];
    for (int x = 0; x < 128; x++) samplebuffer[x] = x;
    GenerateAnalogWaveformRX( 10000, 128, samplebuffer );

    printf( "Generating PWM Waveforms on D0 and D1.\n");
    GeneratePWMWaveformRX( 0, 1000, 50 );
    GeneratePWMWaveformRX( 1, 2000, 25 );

    printf( "Generating Digital Waveform on D2 thru D7.\n");
    LoadDigitalWaveformRX( 128, samplebuffer );
    GenerateDigitalWaveformRX( 0xFC, 1, 1, 0, 0, 0, 0, 3000000);

    printf( "Reading Edge Counts and Frequencies.\n");
    DisableCountsRX();
    ClearCountsRX();
    EnableCountsRX();

    Sleep(100);

    unsigned long ec0, ec1, ec2, ec3, f4, f5, f6, f7;
    GetFrequencyAndCountsRX( &ec0, &ec1, &ec2, &ec3, &f4, &f5, &f6, &f7 );
    printf("Edge Counts 0: %ld\n", ec0);
    printf("Edge Counts 1: %ld\n", ec1);
    printf("Edge Counts 2: %ld\n", ec2);
    printf("Edge Counts 3: %ld\n", ec3);
    printf("Frequency 4: %ld\n", f4);
    printf("Frequency 5: %ld\n", f5);
    printf("Frequency 6: %ld\n", f6);
    printf("Frequency 7: %ld\n", f7);

    printf( "Setting Logic Threshold.\n");
    SetLogicThresholdRX( 1.5 );

    printf( "Starting a Capture.\n");

    // Setup the Trigger Specification
    SetTriggersRX( false, false, false, false, false, false,
    // No trigger, means trigger always
    false, false, false, false, false, false, false, 0, false, 1, false,
    1, 1.0, 3.0, 1.5, 0xFFFF, 0x0000, 0xFFFF, // Trigger X Event
    false, false, false, false, false, false, false, 0, false, 1, false,
    1, 1.0, 3.0, 1.5, 0xFFFF, 0x0000, 0xFFFF); // Trigger Y Event

    // Start the capture
    unsigned long SampleBufferSize = 4*1024*1024; // 4 Million
    unsigned char TriggerPosition = 50; // 50 percent (trigger in the middle)
    float SampleRate = 100000000; // 100MSPS
    unsigned long ChannelMask = 0xFFFFFFFF; // Sample all channels

    StartCaptureRX(SampleBufferSize, TriggerPosition, SampleRate, ChannelMask,
0, 0 );

```



```

// Monitor the status of the capture in progress
unsigned char Full;
unsigned char Triggered;
unsigned char Running = 1;

while( Running )
{
    CaptureStatusRX( &Full, &Triggered, &Running );
    printf("CaptureStatus Full:%d Triggered:%d Running:%d\n", Full,
    Triggered, Running );
    Sleep(10);
}

//TriggerNowRX(); // To trigger now and store data to the end of the buffer
//StopCaptureRX(); // To stop capture in progress and discard the data

// Now that the capture completed, properly end the capture and get the data
__int64 ActualNumberOfSamples;
__int64 TriggerSamplePosition;

// Completes the capture and transfers all of the stored data to the PC
EndCaptureRX( &ActualNumberOfSamples, &TriggerSamplePosition);

printf("Number Of Samples:%I64d Trigger Sample Position:%I64d \n",
ActualNumberOfSamples, TriggerSamplePosition );

// Now access the samples using SampleData() and FindNextEdge() routines
printf("Sample 0 = %04X\n", SampleData(0));
printf("Sample 1 = %04X\n", SampleData(1));
printf("Sample 2 = %04X\n", SampleData(2));
printf("Sample 3 = %04X\n", SampleData(3));
printf("Sample 4 = %04X\n", SampleData(4));

__int64 NextSampleWithEdge = FindNextEdge(1234567, 0xFFFF, 1);

}

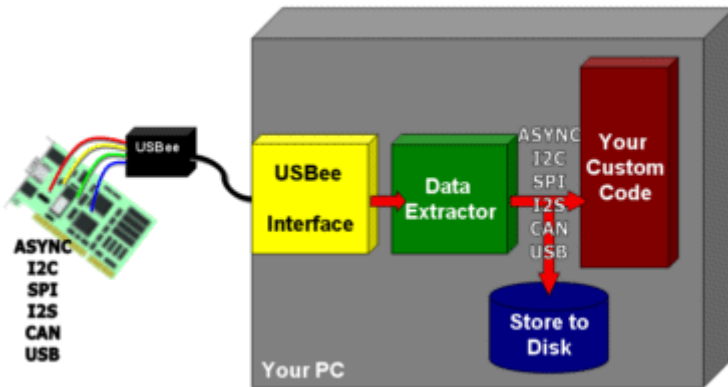
printf( "Press any key to exit.\n");
getch();

return 0;
}

```

## USBEE RX DATA EXTRACTOR OVERVIEW

The Data Extractors are a separate software product for use with the USBee RX Test Pod that allows engineers to extract the raw data from various embedded busses to store off to disk or stream to another application. The Data Extractors will collect the raw data from Parallel, Serial, I2C, I2S, Async, USB Full and Low Speed, SMBus, 1-Wire or CAN busses and store the data to disk or pass it to your own processing application in real-time.



## DATA EXTRACTOR FEATURES

- Uses the USBee RX pod to stream data from your embedded design into your PC
- Captures continuous real-time bus data
- Extracts the transaction data on the fly
- Stores data to disk or process it in real-time
- Runs indefinitely
- Captures entire test sequences
- Monitors embedded system data flows during normal operation
- Processes or stores Megabytes, Gigabytes or Terabytes of data
- Runs as a Windows Command Line executable from the Command Prompt and can be executed from Batch files containing the desired parameters
- Lets you write your own software to further process the extracted data using the Extractor API libraries.

## BUS TYPES DECODED

- **Parallel** (internal or external clocking up to 12MHz)
- **Serial** (internal or external clocking up to 12MHz)
- **Async** (up to 12Mbaud)
- **I2C** (SCL up to 4MHz)
- **SPI** (SPI Clock up to 12MHz)
- **1-Wire** (Standard 1-Wire bit rates)
- **I2S** (bit clock up to 12MHz)
- **USB** (Low 1.5Mbps and Full Speed 12Mbps USB)
- **CAN** (up to 12Mbps)
- **SM Bus** (SM Clock up to 12MHz)

## YOUR TESTING SYSTEM

The typical challenge in embedded streaming bus systems is to get the data out of your embedded system quickly and easily so that you can process it, either to capture a bug in progress or to evaluate performance. In any case, this can be done with the USBee RX Data Extractor System.

The USBee RX pod is used to stream raw sample data from its lowest 8 digital input lines (digital input signal 0 thru 7) directly into the PC. The Data Extractor software modules then take that streaming data and extract your desired data out of the raw stream using the extractor processing threads. Our sample command line application, as well as any custom application you write, interfaces to the extractor through a simple Windows DLL consisting of five function calls. These calls are used to start and extraction, stop an extraction, gather the data (and how much data) and check for error status.

## SYSTEM REQUIREMENTS

- The USBee RX Data Extractors require the following PC configuration:
- Windows® Vista, XP or Windows® 2000 operating system
- Pentium or higher processor
- One USB2.0 High Speed enabled port. It will not run on USB 1.1 Full Speed ports.
- 32MBytes of RAM
- 125MBytes of Hard disk space

It is HIGHLY recommended that the USBee RX and Data Extractors be run together on a separate PC than the PC controlling the system under test. If your PC is also controlling the system under test you may not be able to get the maximum sample rates needed for some of the extractors.

To get the highest sample rates, you will want to use a Desktop PC with native USB 2.0 ports on the motherboard. Some modern Laptops can achieve the maximum of 24MSPs, but you will want to disable all power saving features and run your laptop from the power supply, not the batteries.

## SYSTEM SETUP

To configure a system to run these extractors you need the following:

- USBee RX Suite Software Installed – this installs the USBee Data Extractors Software as well
- USBee RX Pod plugged into a USB 2.0 port on your PC.

---

## RUNNING THE COMMAND LINE EXTRACTORS

Once these components are installed correctly you can run the Extractor command prompt application .exe files. Each of the executables requires a series of command line parameters that tell the extractor how to process the bus data.

You will need to have full security access for the folders that you are running the applications from since they write to these directories for output data. If you do not have access, you will need to either move them or grant yourself access to those directories using the Window Security Settings.

To run the programs, you can do one of two options:

Open a Windows Command Prompt Window, change directory (cd) to your **\Program Files\USBee RX Suite\DataExtractors** directory, and enter the command line including all desired parameters.

or

Edit the batch files (goUSB.bat, gol2C.bat. etc.) in that directory to include the parameters you desire. You can then simply double click on the batch files themselves in the Windows Explorer.

For all of the extractors you will need to use the USBee Pod ID on your Pod (on the back of the unit) as a command line parameter.

---

## BUILDING YOUR OWN PROGRAMS USING THE API

You can also start to build your own processing programs using the source code for the command prompt applications as a reference point. Each Extractor has a sample project (Visual Studio 2010) in the **\Program Files\USBee RX Suite\Data Extractors** directories for you to start with.

## ASYNCR DATA EXTRACTOR

The Async Bus Data Extractor takes the real-time streaming data from up to 8 embedded asynchronous buses (UART), formats it and allows you to save the data to disk or process it as it arrives.

The RX Streaming Data Extractors are optional software modules for use with the USBee RX Test Pod (required) which must be purchased separately.

## ASYNCR BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- 8 digital channels
- Wide range of voltage input (+/- 60V)
- Variable logic Level threshold (0 to 2V)
- Baud Rates from 1200 baud to 12 Mbaud \*
- Data Bit Settings (5, 6, 7 or 8)
- Parity Bit Settings (Mark, Space, Odd, Even, Ignore, None)
- Time Stamps of start of bytes or packets
- Output to Text File (Hex, Decimal, Binary or ASCII)\*
- Output to Screen\*
- Comma, Space, or Newline Delimited files
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

\* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

## HARDWARE SETUP

To use the Data Extractor you need to connect the USBee RX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The Async Bus Data Extractor uses any of the 8 signal lines (0 thru 7) and the GND (ground) line. Connect any of the 8 signal lines to an Async data bus. Connect the GND line to the digital ground of your system.

## EXTRACTOR COMMAND LINE PROGRAM

The Async Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

## To run the Data Extractor:

- Install the USBee RX software on your PC
- Plug in your USBee RX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- Run the executable using the following command line arguments:

```
AsyncExtractor [-?SADHBICGNXYZ] [-R BaudRate] [-E DataBits] [-L Parity] [-M Sign
alMask] [-Q NumberOfBytes] [-V Timestamp] [-O filename] -P PodID

? - Display this help screen

USBee DX-Pro Pod to Use
P - Pod ID (required)

Output Location Flags
O - Output to filename (default off)
S - Output to the screen (default off)

When to Quit Flags
Q - Number of output values (default = until keypress)

Input Format Flags
R - Baud Rate (9600 baud default)
E - Number of Data Bits (5,6,7,8-default)
L - Parity Type (0=none(default), 1=mark, 2=space, 3=even, 4=odd)
M - Which Signals to capture (1=signal0, 128=signal7, 255=all, 0=none (default))
Y - LSBit first (default)
Z - MSBit first
F - Number of Stop Bits (2=1 (default), 3=1.5, 4=2)

Output Number Format Flags
A - ASCII Text Values ("1")
D - Decimal Text Values ("49")
H - Hex Text Values ("31") default
B - Binary Text Values ("00110001")
I - Binary Values (49)
C - Comma Delimited
G - Space Delimited (default)
N - Newline Delimited
X - No Delimiter

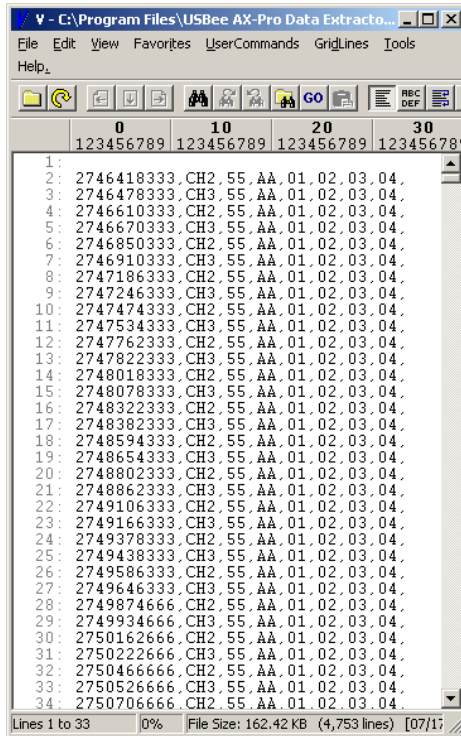
Timestamp and Channel Labels
V - Timestamps and Labels (0=Both off(default),1=Time each byte,2=Time and Labels,3=Labels Only)

Digital Logic Threshold
@ - Logic Threshold * 10 (0=0V, 15=1.5V, etc.)

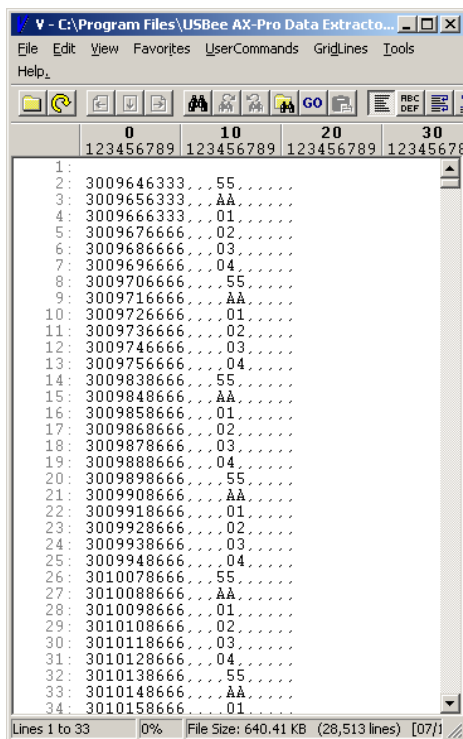
Stop on Overflow
! - Stop on Overflow (default is off)
```

## EXAMPLE OUTPUT FILES

AsyncExtractor -O output.dex -P 3209 -C -Q 100000 -R 1000000 -E 8 -L  
0 -M 255 -H -V 2



```
AsyncExtractor -O output.dex -P 3209 -C -Q 100000 -R 1000000 -E 8 -L
0 -M 255 -H -V 1
```

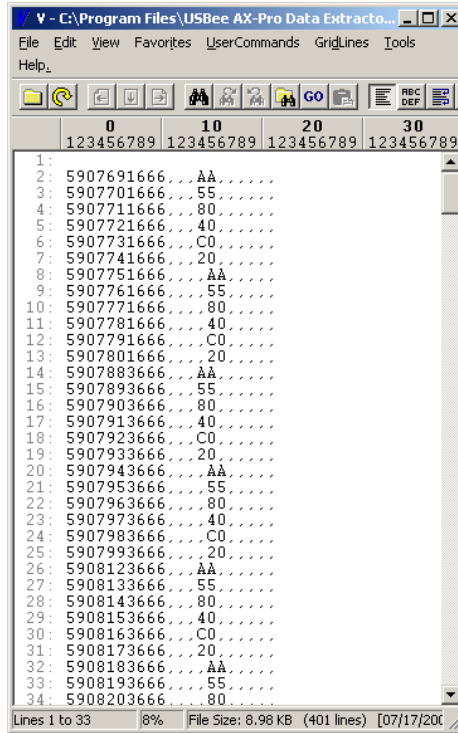




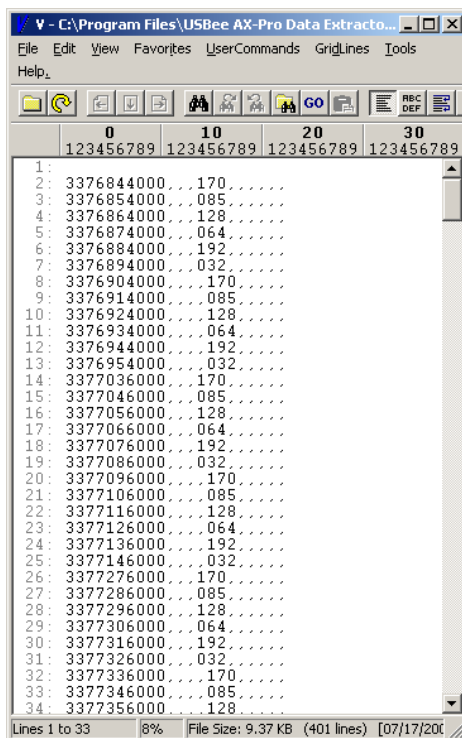
```

AsyncExtractor -S -O output.dex -P 3209 -C -Q 400 -R 1000000 -E 8 -L
0 -M 255 -Z -H -V 1

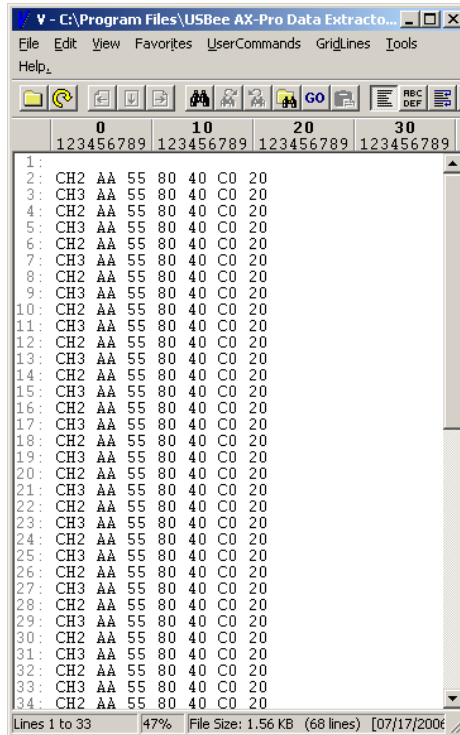
```



```
AsyncExtractor -S -O output.dex -P 3209 -C -Q 400 -R 1000000 -E 8 -L  
0 -M 255 -Z -D -V 1
```



```
AsyncExtractor -S -O output.dex -P 3209 -Q 400 -R 1000000 -E 8 -L 0
-M 255 -Z -H -G -V 3
```



## EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee RX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### DLL FILENAME:

usbedAsync.dll in \Windows\System32

### DLL EXPORTED FUNCTIONS AND PARAMETERS

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using `GetNextData`.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,  
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_IMPORT int CWAV_API StartExtraction(unsigned long  
Threshold, unsigned long PodNumber, unsigned long  
BaudRate, unsigned int DataBits, unsigned int Parity,  
unsigned char Channels, unsigned char MSFirst, unsigned  
char StopBits);
```

Threshold: The voltage above which is a logic 1 and below which is a logic 0 (times 10). 10 = 1.0V, 15 = 1.5V

PodNumber: Pod ID on the back of the USBee RX Test Pod

BaudRate: Baud rate of the async channels. All channels are decoded at the same rate.

Data Bits: Number of Data bits (5, 6, 7 or 8)

Parity:

- 0 = No parity bit
- 1 = Mark Parity
- 2 = Space Parity
- 3 = Even Parity
- 4 = Odd Parity

MSFirst:

- 0 = Least Significant Bit first
- 1 = Most Significant Bit first

Channels: Bit mask for which channels to decode (1 = signal 0, 128 = signal 7)

StopBits:

- 2 = 1 Stop Bit time
- 3 = 1.5 Stop Bit times
- 4 = 2 Stop Bit times

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

CWAV\_EXPORT int CWAV\_API StopExtraction( void );

Returns:

- 1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

CWAV\_EXPORT char CWAV\_API ExtractBufferOverflow(void);

Return:

- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

---

## EXTRACTION DATA FORMAT

The `GetNextData` routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the `*buffer` parameter.

The Async Bus Extractor uses the following format for the data in this buffer:

```
Byte 0:  Timestamp LSByte   (in nanoseconds since start)
Byte 1:  Timestamp
Byte 2:  Timestamp
Byte 3:  Timestamp
Byte 4:  Timestamp
Byte 5:  Timestamp
Byte 6:  Timestamp
Byte 7:  Timestamp MSByte
Byte 8:  Record Type (bit 1 = 1 means character data is valid)
Byte 9:  Channel number (0 thru 7)
Byte 10: Character
Byte 11: Errors (Bit 0 = Parity Error, Bit 1 = Framing (Stop) error)
Byte 12: Control Signal States (all 8 signal bits except async
channels)
Byte 13:  reserved
Byte 14:  reserved
Byte 15:  reserved
(repeat) ...
```

## PARALLEL BUS DATA EXTRACTOR

The Parallel Bus Data Extractor takes the real-time streaming data from an embedded 8-bit parallel bus, formats it and allows you to save the data to disk or process it as it arrives.

### PARALLEL BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- 8 digital channels
- Wide range of voltage input (+/- 60V)
- Variable logic Level threshold (0 to 2V)
- Synchronous or Asynchronous Clocking
- Synchronous (external) clock 0 to 16MB/s\*
- Asynchronous (internal) clock 1MB/s to 24MB/s\*
- Input in 1, 2 or 4 byte serial words
- Little or Big Endian
- Output to Binary File\*
- Output to Text File (Hex, Decimal, Binary or ASCII)\*
- Output to Screen\*
- Comma, Space, or Newline Delimited files
- Output Value Filtering
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

\* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

### HARDWARE SETUP

To use the Data Extractor you need to connect the USBee RX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The Parallel Bus Data Extractor uses the 8 signal lines (0 thru 7), the GND (ground) line and optionally the CLK and TRG lines (for external timing). The signal 0 is represented in the bit 0 of each sampled byte. Connect the GND line to the digital ground of your system.

## EXTRACTOR COMMAND LINE PROGRAM

The Parallel Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee RX software on your PC
- Plug in your USBee RX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- Run the executable using the following command line arguments:

```
BasicExtractor [-?SADHBICGNX124YZ] [-@ LogicThreshold] [-Q NumberOfBytes] [-T By  
tesPesLine] [-R SampleRate] [-M SignalMask] [-L FilterValue] [-V FilterMask] [-O  
filename] -P PodID
```

? - Display this help screen

USBee RX Pod to Use

P - Pod ID (required)

Output Location Flags

O - Output to filename (default off)

S - Output to the screen (default off)

When to Quit Flags

Q - Number of output values (default = until keypress)

Input Number Format Flags

1 - One Byte per value (default)

2 - Two Bytes per value

4 - Four Bytes per value

Y - Least significant byte first

Z - Most significant byte first

Output Number Format Flags

A - ASCII Text Values ("1")

D - Decimal Text Values ("49")

H - Hex Text Values ("31") default



B - Binary Text Values ("00110001")  
I - Binary Values (49)  
C - Comma Delimited  
G - Space Delimited (default)  
N - Newline Delimited  
X - No Delimeter  
T - Force Bytes Per Line (no force default)

#### Filter Values

M - Which Signals to capture (1=signal0,255=all(default))  
L - Filter Mask (0=no filter,255=filter on all signals)  
V - Filter Value (0=store when 0's,255=store when 1's)

#### Clocking Modes

E - Clocking mode (2=internal (default),  
4=CLK rising,5=CLK falling,  
6=CLK rising AND TRG high,7=CLK falling AND TRG high  
8=CLK rising AND TRG low,9=CLK falling AND TRG low  
R - Internal CLK Sample Rate (1Msps default)

#### Digital Logic Threshold

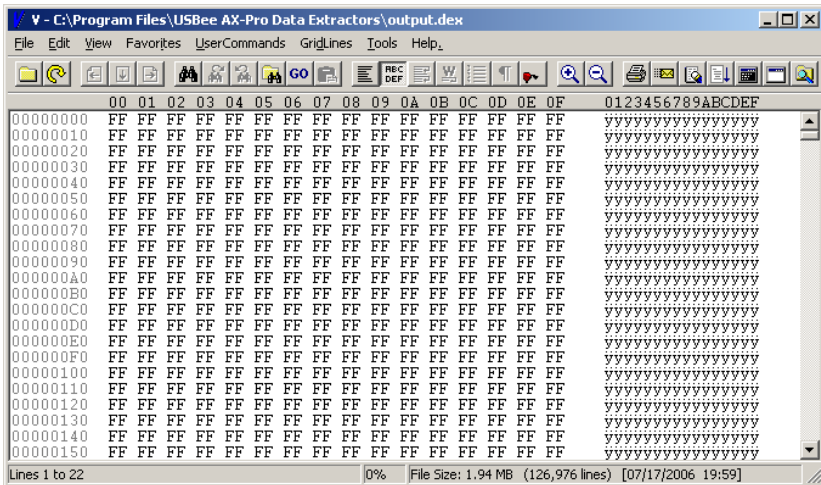
@ - Logic Threshold \* 10 (0=0V, 15=1.5V, etc.)

#### Stop on Overflow

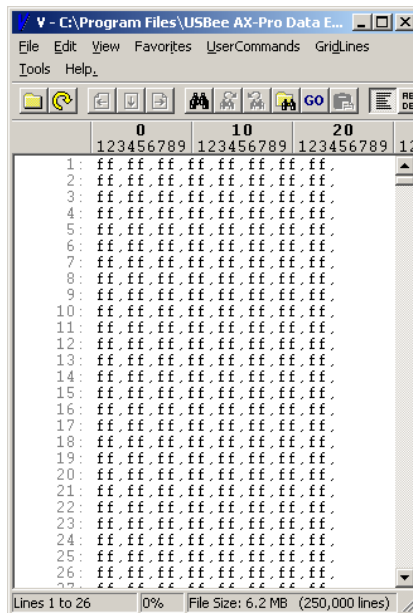
! - Stop on Overflow (default is off)

## EXAMPLE OUTPUT

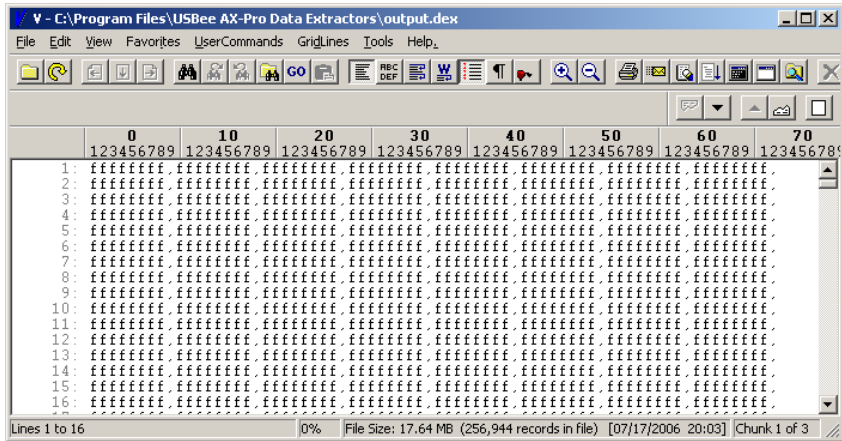
BasicExtractor -O output.dex -P 3209 -1 -R 27 -T 8 -Q 2000000 -I



BasicExtractor -O output.dex -P 3209 -1 -R 27 -T 8 -Q 2000000 -C



```
BasicExtractor -O output.dex -P 3209 -I -R 27 -T 8 -Q 2000000 -C -4
```



## EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee RX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### DLL FILENAME:

usbedBasic.dll in \Windows\System32

### DLL EXPORTED FUNCTIONS AND PARAMETERS

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using `GetNextData`.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,  
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_IMPORT int CWAV_API StartExtraction( unsigned int
SampleRate, unsigned long PodNumber, unsigned int ClockMode,
unsigned long Threshold );
```

Threshold: The voltage above which is a logic 1 and below which is a logic 0 (times 10). 10 = 1.0V, 15 = 1.5V

SampleRate:

- 17 = 1MSPS
- 27 = 2MSPS
- 37 = 3MSPS
- 47 = 4MSPS
- 67 = 6MSPS
- 87 = 8MSPS
- 127 = 12MSPS
- 167 = 16MSPS
- 247 = 24MSPS

PodNumber: Pod ID on the back of the USBee RX Test Pod

ClockMode:

- 2 = Internal Timing as in SampleRate parameter
- 4 – External Timing – sample on rising edge of CLK
- 5 – External Timing – sample on falling edge of CLK
- 6 – External Timing – sample on rising edge of CLK and TRG high
- 7 – External Timing – sample on falling edge of CLK and TRG high
- 8 – External Timing – sample on rising edge of CLK and TRG low
- 9 – External Timing – sample on falling edge of CLK and TRG low

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

---

## EXTRACTION DATA FORMAT

The `GetNextData` routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the `*buffer` parameter.

The Parallel Bus Extractor uses the following format for the data in this buffer:

Byte 0:	Byte 0 of the sampled data
Byte 1:	Byte 1 of the sampled data
Byte 2:	Byte 2 of the sampled data
Byte 3:	Byte 3 of the sampled data
...	
Byte N:	Byte N of the sampled data

## SERIAL BUS DATA EXTRACTOR

The Serial Bus Data Extractor takes the real-time streaming data from up to 8 serial data lines, formats it and allows you to save the data to disk or process it as it arrives.

### SERIAL BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- 8 digital channels
- Synchronous or Asynchronous Clocking
- Synchronous (external) clock 0 to 16MB/s\*
- Asynchronous (internal) clock 1MB/s to 24MB/s\*
- Input in 1, 2 or 4 byte serial words
- Little or Big Endian
- Output to Binary File\*
- Output to Text File (Hex, Decimal, Binary or ASCII)\*
- Output to Screen\*
- Comma, Space, or Newline Delimited files
- Output Value Filtering
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

\* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

### HARDWARE SETUP

To use the Data Extractor you need to connect the USBee RX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The Serial Bus Data Extractor uses any of the 8 signal lines (0 thru 7), the GND (ground) line and optionally the CLK and TRG lines (for external timing). Connect the GND line to the digital ground of your system.

### EXTRACTOR COMMAND LINE PROGRAM

The Serial Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee RX software on your PC

- Plug in your USBee RX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- Run the executable using the following command line arguments:

```
SerialExtractor [-?SADHBICGNX124YZ] [-Q NumberOfBytes] [-T
BytesPerLine] [-R SampleRate] [-E ClockingMode] [-M SignalMask] [-J
ChannelAlign] [-L SignalLevel] [-V AlignmentValue] [-O filename] -P
PodID
```

? - Display this help screen

USBee RX Pod to Use

P - Pod ID (required)

Output Location Flags

O - Output to filename (default off)

S - Output to the screen (default off)

When to Quit Flags

Q - Number of output values (default = until keypress)

Input Number Format Flags

1 - One Byte per value (default)

2 - Two Bytes per value

4 - Four Bytes per value

Y - Least significant byte first

Z - Most significant byte first

Output Number Format Flags

A - ASCII Text Values ("1")

D - Decimal Text Values ("49")

H - Hex Text Values ("31") default

B - Binary Text Values ("00110001")

I - Binary Values (49)

C - Comma Delimited

G - Space Delimited (default)

N - Newline Delimited

X - No Delimiter

T - Force Bytes Per Line (no force default)

Filter Values

M - Which Signals to capture (1=signal0,255=all(default))

Clocking Modes

E - Clocking mode (2=internal (default),

4=CLK rising,5=CLK falling,

6=CLK rising AND TRG high,7=CLK falling AND

TRG high

8=CLK rising AND TRG low,9=CLK falling AND

TRG low

R - Internal CLK Sample Rate (1MSPS default)

Bit Zero Alignment Setting

V - Align on Value

L - Align on Signal Level (0=Low, 1=High)

J - Align on Which Channel (1=Ch 0, 128=Ch 7)



```
Digital Logic Threshold
@ - Logic Threshold * 10 (0=0V, 15=1.5V, etc.)

Stop on Overflow
! - Stop on Overflow (default is off)
```

## EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee RX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### DLL FILENAME:

```
usbSerial.dll in \Windows\System32
```

### DLL EXPORTED FUNCTIONS AND PARAMETERS

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using `GetNextData`.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate,  
unsigned long PodNumber, unsigned int ClockMode, unsigned long  
Threshold, unsigned long AlignValue, unsigned char SignalLevel,  
unsigned char AlignChannel, unsigned char BytePerValue);
```

Threshold: The voltage above which is a logic 1 and below which is a logic 0 (times 10). 10 = 1.0V, 15 = 1.5V

SampleRate:

- 17 = 1Msps
- 27 = 2Msps
- 37 = 3Msps
- 47 = 4Msps
- 67 = 6Msps
- 87 = 8Msps
- 127 = 12Msps
- 167 = 16Msps
- 247 = 24Msps

PodNumber: Pod ID on the back of the USBee RX Test Pod

ClockMode:

- 2 = Internal Timing as in SampleRate parameter
- 4 – External Timing – sample on rising edge of CLK
- 5 – External Timing – sample on falling edge of CLK
- 6 – External Timing – sample on rising edge of CLK and TRG high
- 7 – External Timing – sample on falling edge of CLK and TRG high
- 8 – External Timing – sample on rising edge of CLK and TRG low
- 9 – External Timing – sample on falling edge of CLK and TRG low

AlignValue: Value which the extractor syncs with to define bit 0 alignment.

SignalLevel: Level, 0 or 1, which the extractor syncs with to define bit 0 alignment

AlignChannel: Which signal the extractor uses for alignment, either via value or signal

BytesPerValue: 1, 2, or 4. Used for Value alignment size.

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

---

## EXTRACTION DATA FORMAT

The `GetNextData` routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the `*buffer` parameter.

The Serial Bus Extractor uses the following format for the data in this buffer:

Byte 0:	Channel 0, first byte extracted
Byte 1:	Channel 1, first byte extracted
Byte 2:	Channel 2, first byte extracted
Byte 3:	Channel 3, first byte extracted
Byte 4:	Channel 4, first byte extracted
Byte 5:	Channel 5, first byte extracted
Byte 6:	Channel 6, first byte extracted
Byte 7:	Channel 7, first byte extracted
Byte 8:	Channel 0, second byte extracted
Byte 9:	Channel 1, second byte extracted
...	
Byte N:	Channel (N mod 8), byte (N/8)+1 extracted

## I2C DATA EXTRACTOR

The I<sup>2</sup>C Bus Data Extractor takes the real-time streaming data from the I2C bus, formats it and allows you to save the data to disk or process it as it arrives.

### I2C DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- Monitors one I<sup>2</sup>C Bus
- Time Stamp for each packet
- Output to Text File\*
- Output to Screen\*
- Comma or Space Delimited files
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

\* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

### HARDWARE SETUP

To use the Data Extractor you need to connect the USBee RX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The I<sup>2</sup>C Bus Data Extractor connects to the SDA and SCL lines of the I<sup>2</sup>C bus. Use one signal as the SDA data line and one signal as the SCL clock line. Also connect the GND line to the digital ground of your system. Connect these signals to the I<sup>2</sup>C bus using the test clips provided.

### EXTRACTOR COMMAND LINE PROGRAM

The I<sup>2</sup>C Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee RX software on your PC
- Plug in your USBee RX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- Run the executable using the following command line arguments:

```
I2CExtractor [-?SDHICGAB] [-Q NumberOfBytes] [-V Timestamp] [-O  
filename] [-M SDAMask] [-N SCLMask] -P PodID
```

```
? - Display this help screen
```

USBee RX Pod to Use

```
P - Pod ID (required)
```

Output Location Flags

```
O - Output to filename (default off)  
S - Output to the screen (default off)
```

When to Quit Flags

```
Q - Number of output values (default = until keypress)
```

Input Flags

```
M - SDA signal (1=ch0, 128=ch7, ch0 default)  
N - SCL signal (1=ch0, 128=ch7, ch1 default)
```

Output Number Format Flags

```
A - All Packet Fields are output (default)  
B - Only data bytes are output  
D - Decimal Text Values ("49")  
H - Hex Text Values ("31") default  
I - Binary Values (49)  
C - Comma Delimited  
G - Space Delimited (default)  
V - Timestamps (0=off(default),1=Timestamp on
```

Digital Logic Threshold

```
@ - Logic Threshold * 10 (0=0V, 15=1.5V, etc.)
```

Stop on Overflow

```
! - Stop on Overflow (default is off)
```

## EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee RX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

---

### DLL FILENAME:

usbedI2C.dll in \Windows\System32

---

## DLL EXPORTED FUNCTIONS AND PARAMETERS

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using `GetNextData`.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,  
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_IMPORT int CWAV_API StartExtraction( unsigned long Threshold,  
unsigned long PodNumber, unsigned char All, unsigned char Decimal,  
unsigned char Hex, unsigned char Binary, unsigned char Comma,  
unsigned char Space, unsigned char Timestamps, unsigned long  
SDA,unsigned long SCL);
```

Threshold: The voltage above which is a logic 1 and below which is a logic 0 (times 10). 10 = 1.0V, 15 = 1.5V

PodNumber: Pod ID on the back of the USBee RX Test Pod

All:

- 0 – Only the data payload bytes are returned
- 1 – All I2C packet fields are returned

Decimal:

- 1 – Decimal Values (text) are output for the data bytes

Hex:

- 1 – Hex Values (text) are output for the data bytes

Binary:

- 1 – All data is in binary form, not text

Comma:

- 1 – Commas are placed between each field/data byte

Space:

- 1 – Spaces are placed between each field/data byte

Timestamp:

- 1 – Print Timestamps at the start of each packet

SDAMask:

- The mask for the channel to use for SDA
- (1 = Ch0, 128 = Ch7)

SCLMask:

- The mask for the channel to use for SCL
- (1 = Ch0, 128 = Ch7)

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

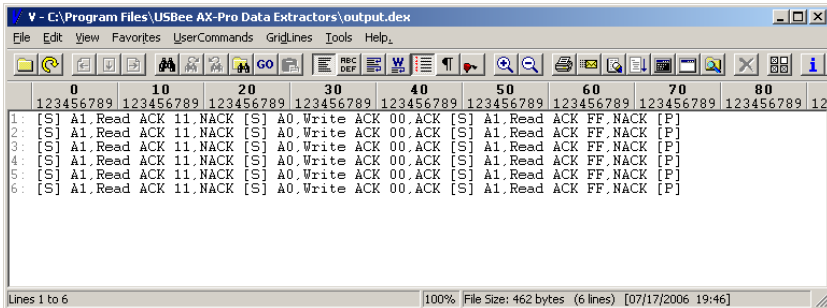
- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

## EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the \*buffer parameter.

The I<sup>2</sup>C Bus Extractor DLL sends the extracted data through the \*buffer in the requested form based on the parameters in the StartExtraction call. For example, if Binary is set to a 0, then the \*buffer will receive the binary bytes that make up the data stream. If Hex is set to a 1, the \*buffer will contain a text string which is the data of the I2C traffic in Hex text form, separated by any specified delimiters.

```
I2CExtractor -O output.dex -P 3209 -Q 5000 -H -C -M 2 -N 1 -V 0
```







## SM BUS DATA EXTRACTOR

The SM Bus Data Extractor takes the real-time streaming data from the SM bus, formats it and allows you to save the data to disk or process it as it arrives.

### SM BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- Monitors one SM Bus
- Time Stamp for each packet
- Output to Text File\*
- Output to Screen\*
- Comma or Space Delimited files
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

\* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

### HARDWARE SETUP

To use the Data Extractor you need to connect the USBee RX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The SM Bus Data Extractor connects to the SMBClk and SMBData lines of the SM Bus. Use one signal as the SMBData line and one signal as the SMBClk line. Also connect the GND line to the digital ground of your system. Connect these signals to the SM Bus using the test clips provided.

### EXTRACTOR COMMAND LINE PROGRAM

The SM Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee RX software on your PC
- Plug in your USBee RX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- Run the executable using the following command line arguments:

```
SMBusExtractor [-?SDHICGAB] [-Q NumberOfBytes] [-V Timestamp] [-O  
filename] [-M SMBDatMask] [-N SMBClkMask] -P PodID
```

```
? - Display this help screen
```

USBee RX Pod to Use

```
P - Pod ID (required)
```

Output Location Flags

```
O - Output to filename (default off)
```

```
S - Output to the screen (default off)
```

When to Quit Flags

```
Q - Number of output values (default = until keypress)
```

Output Number Format Flags

```
A - All Packet Fields are output (default)
```

```
B - Only data bytes are output
```

```
D - Decimal Text Values ("49")
```

```
H - Hex Text Values ("31") default
```

```
I - Binary Values (49)
```

```
C - Comma Delimited
```

```
G - Space Delimited (default)
```

```
V - Timestamps (0=off(default),1=Timestamp on
```

```
M - SMBData signal (1=ch0, 128=ch7, ch0 default)
```

```
N - SMBClk signal (1=ch0, 128=ch7, ch1 default)
```

Digital Logic Threshold

```
@ - Logic Threshold * 10 (0=0V, 15=1.5V, etc.)
```

Stop on Overflow

```
! - Stop on Overflow (default is off)
```

## EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee RX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

---

## DLL FILENAME:

usbSMBus.dll in \Windows\System32

---

## DLL EXPORTED FUNCTIONS AND PARAMETERS

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using **GetNextData**.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,  
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_IMPORT int CWAV_API StartExtraction(unsigned long Threshold ,  
unsigned long PodNumber, unsigned char All, unsigned char Decimal,  
unsigned char Hex, unsigned char Binary, unsigned char Comma,  
unsigned char Space, unsigned char Timestamps, unsigned long  
SMBData,unsigned long SMBClk);
```

Threshold: The voltage above which is a logic 1 and below which is a logic 0 (times 10). 10 = 1.0V, 15 = 1.5V

PodNumber: Pod ID on the back of the USBee RX Test Pod

All:

- 0 – Only the data payload bytes are returned
- 1 – All SMBus packet fields are returned

Decimal:

- 1 – Decimal Values (text) are output for the data bytes

Hex:

- 1 – Hex Values (text) are output for the data bytes

Binary:

- 1 – All data is in binary form, not text

Comma:

- 1 – Commas are placed between each field/data byte

Space:

- 1 – Spaces are placed between each field/data byte

Timestamp:

- 1 – Print Timestamps at the start of each packet

SMBData:

- The mask for the channel to use for Data
- (1 = Ch0, 128 = Ch7)

SMDClk:

- The mask for the channel to use for Clk
- (1 = Ch0, 128 = Ch7)

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

---

## EXTRACTION DATA FORMAT

The `GetNextData` routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the `*buffer` parameter.

The SM Bus Extractor DLL sends the extracted data through the `*buffer` in the requested form based on the parameters in the `StartExtraction` call. For example, if `Binary` is set to a 0, then the `*buffer` will receive the binary bytes that make up the data stream. If `Hex` is set to a 1, the `*buffer` will contain a text string which is the data of the SMBus traffic in Hex text form, separated by any specified delimiters.

Line	Address	Status	Description
1:	0000001832	[S]	A1 Read 46 NACK [S]
2:	0000054106	[S]	A0 Write 00 11 [P]
3:	0000054106	[S]	A0 Write NACK [P]
4:	0000054106	[S]	A0 Write NACK [P]
5:	0000054106	[S]	A0 Write NACK [P]
6:	0000054106	[S]	A0 Write NACK [P]
7:	0000054107	[S]	A0 Write NACK [P]
8:	0000054107	[S]	A0 Write NACK [P]
9:	0000054107	[S]	A0 Write NACK [P]
10:	0000054107	[S]	A0 Write NACK [P]
11:	0000054107	[S]	A0 Write NACK [P]
12:	0000054107	[S]	A0 Write NACK [P]
13:	0000054107	[S]	A0 Write NACK [P]
14:	0000054107	[S]	A0 Write NACK [P]
15:	0000054108	[S]	A0 Write NACK [P]
16:	0000054108	[S]	A0 Write [P]
17:	0000054108	[S]	A0 Write 01 22 [P]
18:	0000054108	[S]	A0 Write NACK [P]
19:	0000054108	[S]	A0 Write NACK [P]
20:	0000054109	[S]	A0 Write NACK [P]
21:	0000054109	[S]	A0 Write NACK [P]
22:	0000054109	[S]	A0 Write NACK [P]
23:	0000054109	[S]	A0 Write NACK [P]
24:	0000054109	[S]	A0 Write NACK [P]
25:	0000054109	[S]	A0 Write NACK [P]
26:	0000054109	[S]	A0 Write NACK [P]
27:	0000054110	[S]	A0 Write NACK [P]
28:	0000054110	[S]	A0 Write NACK [P]
29:	0000054110	[S]	A0 Write NACK [P]
30:	0000054110	[S]	A0 Write NACK [P]
31:	0000054110	[S]	A0 Write [P]
32:	0000054110	[S]	A0 Write 02 33 [P]
33:	0000054111	[S]	A0 Write NACK [P]
34:	0000054111	[S]	A0 Write NACK [P]
35:	0000054111	[S]	A0 Write NACK [P]
36:	0000054111	[S]	A0 Write NACK [P]
37:	0000054111	[S]	A0 Write NACK [P]
38:	0000054111	[S]	A0 Write NACK [P]
39:	0000054111	[S]	A0 Write NACK [P]
40:	0000054112	[S]	A0 Write NACK [P]
41:	0000054112	[S]	A0 Write NACK [P]
42:	0000054112	[S]	A0 Write NACK [P]
43:	0000054112	[S]	A0 Write NACK [P]
44:	0000054112	[S]	A0 Write NACK [P]
45:	0000054112	[S]	A0 Write NACK [P]
46:	0000054112	[S]	A0 Write [P]
47:	0000054113	[S]	A0 Write 03 44 [P]
48:	0000054113	[S]	A0 Write NACK [P]
49:	0000054113	[S]	A0 Write NACK [P]
50:	0000054113	[S]	A0 Write NACK [P]
51:	0000054113	[S]	A0 Write NACK [P]
52:	0000054114	[S]	A0 Write NACK [P]
53:	0000054114	[S]	A0 Write NACK [P]
54:	0000054114	[S]	A0 Write NACK [P]
55:	0000054114	[S]	A0 Write NACK [P]
56:	0000054114	[S]	A0 Write NACK [P]
57:	0000054114	[S]	A0 Write NACK [P]
58:	0000054114	[S]	A0 Write NACK [P]
59:	0000054115	[S]	A0 Write NACK [P]
60:	0000054115	[S]	A0 Write NACK [P]
61:	0000054115	[S]	A0 Write [P]
62:	0000054115	[S]	A0 Write 04 55 [P]
63:	0000054115	[S]	A0 Write NACK [P]
64:	0000054116	[S]	A0 Write NACK [P]
65:	nnnnnn54116	[S]	A0 Write NACK [P]

Lines 1 to 64 | 26% | File Size: 7.98 KB (241 lines) | [07/12/2006 00:33]

## SPI DATA EXTRACTOR

The SPI Bus Data Extractor takes the real-time streaming data from an SPI bus, formats it and allows you to save the data to disk or process it as it arrives.

## SERIAL BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- Monitors one SPI Bus
- SPI Clock speeds up to 12MHz
- Asynchronous (internal) sampling of 1MB/s to 24MB/s\*
- Output to Binary File\*
- Output to Text File\*
- Output to Screen\*
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

\* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

## HARDWARE SETUP

To use the Data Extractor you need to connect the USBee RX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The SPI Bus Data Extractor uses any of the 8 signal lines (0 thru 7) and the GND (ground) line. Connect any of the 8 signals lines to Slave Select, MOSI, and MISO. Connect the GND line to the digital ground of your system.

## EXTRACTOR COMMAND LINE PROGRAM

The SPI Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.



To run the Data Extractor:

- Install the USBee RX software on your PC
- Plug in your USBee RX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- Run the executable using the following command line arguments:

```
SPIExtractor [-?SWT] [-Q NumberOfBytes] [-R SampleRate] [-M  
SlaveSelect] [-L CLK] [-V MOSI] [-J MISO] [-K MOSISample] [-U  
MOSISample] [-O filename] -P PodID
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

M - Slave Select Signal (1=signal0,128=signal7)

L - Clk Signal (1=signal0,128=signal7)

V - MOSI Signal (1=signal0,128=signal7)

J - MISO Signal (1=signal0,128=signal7)

K - MOSI Sample Time (1=Rising CLK Edge,0=Falling CLK Edge)

U - MISO Sample Time (1=Rising CLK Edge,0=Falling CLK Edge)

W - Insert Slave Select Boundaries

T - Insert Time Stamps

R - Internal CLK Sample Rate (16MSPS default)

- 247 = 24MHz
- 167 = 16MHz (default)
- 127 = 12MHz
- 87 = 8MHz
- 67 = 6MHz
- 47 = 4MHz
- 37 = 3MHz
- 27 = 2MHz

- 17 = 1MHz

## EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee RX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

### DLL FILENAME:

usbedSPI.dll in \Windows\System32

### DLL EXPORTED FUNCTIONS AND PARAMETERS

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using `GetNextData`.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate,
unsigned long PodNumber, unsigned int ClockMode, unsigned long
Threshold, unsigned char SlaveSelect, unsigned char CLK, unsigned
char MOSI, unsigned char MISO, unsigned char MOSIEdge, unsigned char
MISOEdge, unsigned char SSInsert, unsigned char Timestamp );
```

Threshold: The voltage above which is a logic 1 and below which is a logic 0 (times 10). 10 = 1.0V, 15 = 1.5V

SampleRate:

- 17 = 1Msps
- 27 = 2Msps
- 37 = 3Msps
- 47 = 4Msps
- 67 = 6Msps
- 87 = 8Msps
- 127 = 12Msps
- 167 = 16Msps
- 247 = 24Msps

PodNumber: Pod ID on the back of the USBee RX Test Pod

ClockMode: 2 = Internal Timing as in SampleRate parameter

SlaveSelect: Which signal the extractor uses for Slave Select (1=channel0,128=channel7)

CLK: Which signal the extractor uses for CLK (1=channel0,128=channel7)

MOSI: Which signal the extractor uses for MOSI (1=channel0,128=channel7)

MISO: Which signal the extractor uses for MISO (1=channel0,128=channel7)

MOSIEdge: When the MOSI signal is sampled, 0=Falling CLK Edge, 1=Rising CLK Edge

MISOEdge: When the MISO signal is sampled, 0=Falling CLK Edge, 1=Rising CLK Edge

SSInsert: Set to 1 to insert Slave Select boundaries into the extracted data stream

Timestamp: Set to 1 to insert Time Stamps into the extracted data stream

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

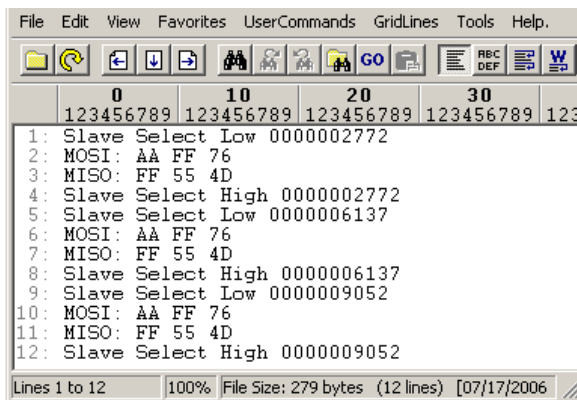
- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

## EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the \*buffer parameter.

The SPI Bus Extractor outputs MOSI and MISO values separated by newline characters with optional Slave Select and Timestamps inserted.

```
SPIExtractor -O output.dex -P 143 -Q 500000 -M 8 -L 1 -V 2 -J 4 -K 1  
-U 0 -W -T
```



## 1-WIRE DATA EXTRACTOR

The 1-Wire Bus Data Extractor takes the real-time streaming data from an 1-Wire bus, formats it and allows you to save the data to disk or process it as it arrives.

### 1-WIRE BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- Monitors one 1-Wire Bus
- Asynchronous (internal) sampling from 1MB/s to 24MB/s\*
- Output to Binary File\*
- Output to Text File\*
- Output to Screen\*
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

\* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

### HARDWARE SETUP

To use the Data Extractor you need to connect the USBee RX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The 1-Wire Bus Data Extractor uses any of the 8 signal lines (0 thru 7) and the GND (ground) line. Connect any of the 8 signals lines to the 1-Wire Signal. Connect the GND line to the digital ground of your system.

### EXTRACTOR COMMAND LINE PROGRAM

The 1-Wire Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee RX software on your PC
- Plug in your USBee RX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- Run the executable using the following command line arguments:

```
1WireExtractor [-?STW] [-Q NumberOfBytes] [-R SampleRate] [-M  
Signal] [-O filename] -P PodID
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

M - 1 Wire Signal Mask (1=channel0,128=channel7)

W - Insert Reset/Presence Pulse

T - Insert Time Stamps

R - Internal CLK Sample Rate (16Msps default)

- 247 = 24MHz
- 167 = 16MHz
- 127 = 12MHz
- 87 = 8MHz
- 67 = 6MHz
- 47 = 4MHz
- 37 = 3MHz
- 27 = 2MHz
- 17 = 1MHz (default)

## EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee RX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

---

### DLL FILENAME:

---

## DLL EXPORTED FUNCTIONS AND PARAMETERS

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using `GetNextData`.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,  
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate,  
unsigned long PodNumber, unsigned int ClockMode, unsigned long  
Threshold, unsigned char Signal, unsigned char SSInsert, unsigned  
char Timestamp );
```

Threshold: The voltage above which is a logic 1 and below which is a logic 0 (times 10). 10 = 1.0V, 15 = 1.5V

SampleRate:

- 17 = 1Msps
- 27 = 2Msps
- 37 = 3Msps
- 47 = 4Msps
- 67 = 6Msps
- 87 = 8Msps
- 127 = 12Msps
- 167 = 16Msps

- 247 = 24Msps

PodNumber: Pod ID on the back of the USBee RX Test Pod

ClockMode: 2 = Internal Timing as in SampleRate parameter

Signal: Which signal the extractor uses for the 1-Wire Signal (1=channel0,128=channel7)

SSInsert: Set to 1 to insert Reset/Presence boundaries into the extracted data stream

Timestamp: Set to 1 to insert Time Stamps into the extracted data stream

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

---

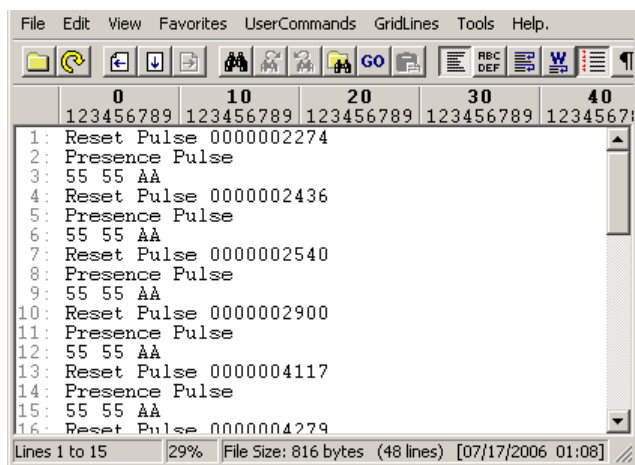
## EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the \*buffer parameter.

The 1-Wire Bus Extractor outputs data values separated by newline characters with option Reset/Presence and Timestamps inserted.



```
1WireExtractor -O output.dex -P 143 -Q 500000 -M 1 -W -T -R 127
```



## I2S DATA EXTRACTOR

The I2S Bus Data Extractor takes the real-time streaming data from an I2S bus, formats it and allows you to save the data to disk or process it as it arrives.

### I2S BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- Monitors one I2S Bus
- I2S Bit Clock up to 12MHz
- Supports I2S or Left Justified sample formats
- Supports MSBit first and non-standard LSBit first formats
- Asynchronous (internal) sampling from 1MB/s to 24MB/s\*
- Output to Binary File\*
- Output to Text File\*
- Output to Screen\*
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

\* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

### HARDWARE SETUP

To use the Data Extractor you need to connect the USBee RX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The I2S Bus Data Extractor uses any of the 8 signal lines (0 thru 7) and the GND (ground) line. Connect any of the 8 signals lines to Word Select, CLK, and Data. Connect the GND line to the digital ground of your system.

### EXTRACTOR COMMAND LINE PROGRAM

The I2S Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee RX software on your PC
- Plug in your USBee RX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.

- Change the working directory to the Data Extractor directory
- Run the executable using the following command line arguments:

```
I2SExtractor [-?ST1234JIYZ] [-Q NumberOfBytes] [-R SampleRate] [-M WordSelect] [-L CLK] [-V Data] [-O filename] -P PodID
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

M - Word Select Signal (1=signal0,128=signal7)

L - Clk Signal (1=signal0,128=signal7)

V - Data Signal (1=signal0,128=signal7)

Y - Least significant bit first

Z - Most significant bit first

J - Left Justified (first rising edge after Word Select change is first bit)

I - I2S format (second rising edge after Word Select change is first bit)

T - Insert Word Select Boundaries

R - Internal CLK Sample Rate (16Msps default)

- 247 = 24MHz
- 167 = 16MHz (default)
- 127 = 12MHz
- 87 = 8MHz
- 67 = 6MHz
- 47 = 4MHz
- 37 = 3MHz
- 27 = 2MHz
- 17 = 1MHz

## EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee RX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

---

## DLL FILENAME:

usbedI2S.dll in \Windows\System32

---

## DLL EXPORTED FUNCTIONS AND PARAMETERS

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using `GetNextData`.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,  
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate,  
unsigned long PodNumber, unsigned int ClockMode, unsigned long  
Threshold, unsigned char WordSelect, unsigned char CLK, unsigned  
char Data, unsigned char SSInsert, unsigned char BytesPerValue,  
unsigned char I2SMode, unsigned char MSBFirstMode );
```

Threshold: The voltage above which is a logic 1 and below which is a logic 0 (times 10). 10 = 1.0V, 15 = 1.5V

SampleRate:

- 17 = 1MSPS

- 27 = 2Msps
- 37 = 3Msps
- 47 = 4Msps
- 67 = 6Msps
- 87 = 8Msps
- 127 = 12Msps
- 167 = 16Msps
- 247 = 24Msps

PodNumber: Pod ID on the back of the USBee RX Test Pod

ClockMode: 2 = Internal Timing as in SampleRate parameter

WordSelect: Which signal the extractor uses for Word Select (1=channel0,128=channel7)

CLK: Which signal the extractor uses for CLK (1=channel0,128=channel7)

Data: Which signal the extractor uses for Data (1=channel0,128=channel7)

SSInsert: Set to 1 to insert Word Select boundaries into the extracted data stream

BytesPerValue: 1, 2, 3, or 4 bytes per value. Allows capture of 8, 16, 24, or 32 bits of audio data

I2SMode: Set to 1 for I2S data format. Set to 0 for Left Justified data format.

MSBFirstMode: Bit order (1 = MSBit first on the wire, 0 = LSBit first on the wire)

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

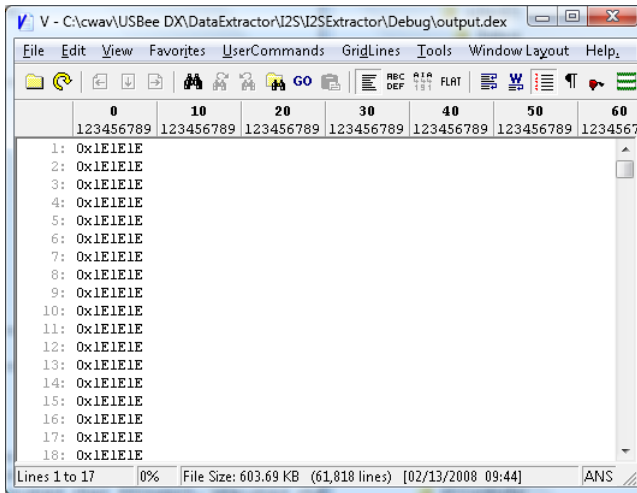
Return:

- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

## EXTRACTION DATA FORMAT

The `GetNextData` routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the `*buffer` parameter.

```
I2SExtractor -O output.dex -P 123 -M 1 -L 2 -V 4 -3 -I
```



## LOW AND FULL SPEED USB DATA EXTRACTOR

The USB Data Extractor takes the real-time streaming data from the Full or Low Speed bus, formats it and allows you to save the data to disk or process it as it arrives.

### USB DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- One USB Bus running at Low (1.5Mbps) or Full Speed (12Mbps) USB (not High Speed)
- Time Stamp for each packet
- Output to Text File\*
- Output to Screen\*
- Comma, Space, or Newline Delimited files
- Packet filter on Device Address, and/or Endpoint
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

\* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

### HARDWARE SETUP

To use the Data Extractor you need to connect the USBee RX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The USB Bus Data Extractor uses signal 0 and signal 1 as the DPlus and DMinus lines of the USB bus. Connect these signals to the USB bus using the test clips provided. Connect the GND line to the digital ground of your system.

### EXTRACTOR COMMAND LINE PROGRAM

The USB Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee RX software on your PC
- Plug in your USBee RX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- Run the executable using the following command line arguments:

```
Usbedtractor [-?SDHICGAB] [-R USBSpeed] [-Q NumberOfBytes] [-V  
Timestamp] [-O filename] -P PodID
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

R - Bus Speed (0=Low Speed USB, 1=Full Speed USB)

A - All Packet Fields are output (default)

B - Only Data Bytes are output

D - Decimal Text Values ("49")

H - Hex Text Values ("31") default

I - Binary Values (49)

C - Comma Delimited

G - Space Delimited (default)

V - Timestamps (0=off, 1=each packet start)

## EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee RX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

---

### DLL FILENAME:

usbedUSB.dll in \Windows\System32

---

### DLL EXPORTED FUNCTIONS AND PARAMETERS

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using `GetNextData`.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```



Returns:

- 0 – No data to read yet
- other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,  
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_IMPORT int CWAV_API StartExtraction(unsigned long Threshold,  
unsigned long PodNumber, unsigned char Speed, unsigned char All,  
unsigned char Decimal, unsigned char Hex, unsigned char Binary,  
unsigned char Comma, unsigned char Space, unsigned char Timestamps,  
unsigned int Endpoint, unsigned int Device);
```

Threshold: The voltage above which is a logic 1 and below which is a logic 0 (times 10). 10 = 1.0V, 15 = 1.5V

PodNumber: Pod ID on the back of the USBee RX Test Pod

Speed:

- 0 = Low Speed
- 1 = Full Speed

All:

- 0 – Only the data payload bytes are returned
- 1 – All USB packet fields are returned

Decimal:

- 1 – Decimal Values (text) are output for the data bytes

Hex:

- 1 – Hex Values (text) are output for the data bytes

Binary:

- 1 – All data is in binary form, not text

Comma:

- 1 – Commas are placed between each field/data byte

Space:

- 1 – Spaces are placed between each field/data byte

Timestamp:

- 1 – Print Timestamps at the start of each packet

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

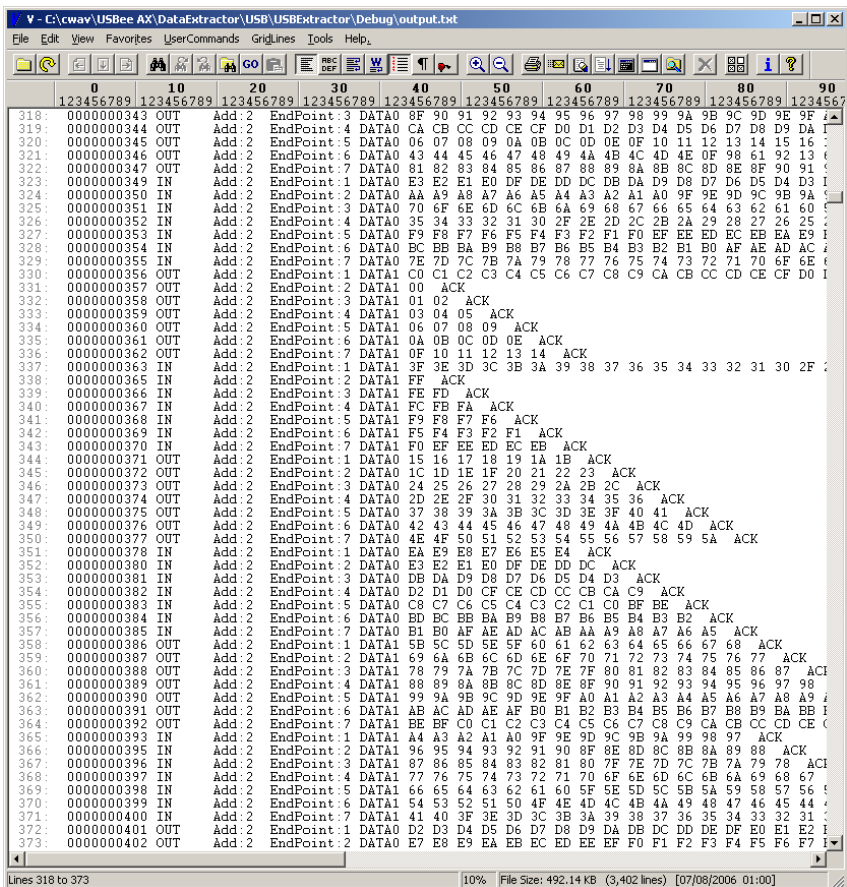
Return:

- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

## EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the \*buffer parameter.

The USB Bus Extractor DLL sends the extracted data through the \*buffer in the requested form based on the parameters in the StartExtraction call. For example, if Binary is set to a 0, then the \*buffer will receive the binary bytes that make up the data stream. If Hex is set to a 1, the \*buffer will contain a text string which is the data of the USB traffic in Hex text form, separated by any specified delimiters.



The screenshot shows a window titled "V: C:\cwav\USBee AX\DataExtractor\USB\USBExtractor\_Debug\output.txt". The window contains a list of data points, each with a line number, a hex value, a direction (IN/OUT), an endpoint, and a data point number. The data points are organized into columns, with headers 0, 10, 20, 30, 40, 50, 60, 70, 80, 90. The data points are as follows:

Line	Hex	Dir	Endpoint	Data Point
318	000000343	OUT	Add:2 EndPoint:3	DATA0 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
319	000000344	OUT	Add:2 EndPoint:4	DATA0 CA FB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA
320	000000345	OUT	Add:2 EndPoint:5	DATA0 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16
321	000000346	OUT	Add:2 EndPoint:6	DATA0 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52
322	000000347	OUT	Add:2 EndPoint:7	DATA0 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91
323	000000349	IN	Add:2 EndPoint:1	DATA0 E3 E2 E1 E0 DF DE DD DC DB DA D9 D8 D7 D6 D5 D4 D3 D1
324	000000350	IN	Add:2 EndPoint:2	DATA0 AA A9 A8 A7 A6 A5 A4 A3 A2 A1 A0 9F 9E 9D 9C 9B 9A 99
325	000000351	IN	Add:2 EndPoint:3	DATA0 70 6F 6E 6D 6C 6B 6A 69 68 67 66 65 64 63 62 61 60 5F
326	000000352	IN	Add:2 EndPoint:4	DATA0 35 34 33 32 31 30 2F 2E 2D 2C 2B 2A 29 28 27 26 25 24
327	000000353	IN	Add:2 EndPoint:5	DATA0 F9 F8 F7 F6 F5 F4 F3 F2 F1 F0 EF EE ED EC EB EA E9 E8
328	000000354	IN	Add:2 EndPoint:6	DATA0 BC BB BA B9 B8 B7 B6 B5 B4 B3 B2 B1 B0 AF AE AD AC AB
329	000000355	IN	Add:2 EndPoint:7	DATA0 7E 7D 7C 7B 7A 79 78 77 76 75 74 73 72 71 70 6F 6E 6D
330	000000356	OUT	Add:2 EndPoint:1	DATA1 C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF DO I
331	000000357	OUT	Add:2 EndPoint:2	DATA1 00 ACK
332	000000358	OUT	Add:2 EndPoint:3	DATA1 01 02 ACK
333	000000359	OUT	Add:2 EndPoint:4	DATA1 03 04 05 ACK
334	000000360	OUT	Add:2 EndPoint:5	DATA1 06 07 08 09 ACK
335	000000361	OUT	Add:2 EndPoint:6	DATA1 0A 0B 0C 0D 0E ACK
336	000000362	OUT	Add:2 EndPoint:7	DATA1 0F 10 11 12 13 14 ACK
337	000000363	IN	Add:2 EndPoint:1	DATA1 3F 3E 3D 3C 3B 3A 39 38 37 36 35 34 33 32 31 30 2F
338	000000365	IN	Add:2 EndPoint:2	DATA1 FF ACK
339	000000366	IN	Add:2 EndPoint:3	DATA1 FE FD ACK
340	000000367	IN	Add:2 EndPoint:4	DATA1 FC FB FA ACK
341	000000368	IN	Add:2 EndPoint:5	DATA1 F9 F8 F7 F6 ACK
342	000000369	IN	Add:2 EndPoint:6	DATA1 F5 F4 F3 F2 F1 ACK
343	000000370	IN	Add:2 EndPoint:7	DATA1 F0 EF EE ED EC EB ACK
344	000000371	OUT	Add:2 EndPoint:1	DATA0 15 16 17 18 19 1A 1B ACK
345	000000372	OUT	Add:2 EndPoint:2	DATA0 1C 1D 1E 1F 20 21 22 23 ACK
346	000000373	OUT	Add:2 EndPoint:3	DATA0 24 25 26 27 28 29 2A 2B 2C ACK
347	000000374	OUT	Add:2 EndPoint:4	DATA0 2D 2E 2F 30 31 32 33 34 35 36 ACK
348	000000375	OUT	Add:2 EndPoint:5	DATA0 37 38 39 3A 3B 3C 3D 3E 3F 40 41 ACK
349	000000376	OUT	Add:2 EndPoint:6	DATA0 42 43 44 45 46 47 48 49 4A 4B 4C 4D ACK
350	000000377	OUT	Add:2 EndPoint:7	DATA0 4E 4F 50 51 52 53 54 55 56 57 58 59 5A ACK
351	000000378	IN	Add:2 EndPoint:1	DATA0 EA E9 E8 E7 E6 E5 E4 ACK
352	000000380	IN	Add:2 EndPoint:2	DATA0 E3 E2 E1 E0 DF DE DD DC ACK
353	000000381	IN	Add:2 EndPoint:3	DATA0 DB DA D9 D8 D7 D6 D5 D4 D3 ACK
354	000000382	IN	Add:2 EndPoint:4	DATA0 D2 D1 D0 CF CE CD CC CB CA C9 ACK
355	000000383	IN	Add:2 EndPoint:5	DATA0 C8 C7 C6 C5 C4 C3 C2 C1 C0 BF BE ACK
356	000000384	IN	Add:2 EndPoint:6	DATA0 BD BC BB BA B9 B8 B7 B6 B5 B4 B3 B2 ACK
357	000000385	IN	Add:2 EndPoint:7	DATA0 B1 B0 AF AE AD AC AB A9 A8 A7 A6 A5 ACK
358	000000386	OUT	Add:2 EndPoint:1	DATA1 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68 ACK
359	000000387	OUT	Add:2 EndPoint:2	DATA1 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 ACK
360	000000388	OUT	Add:2 EndPoint:3	DATA1 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 85 86 87 ACK
361	000000389	OUT	Add:2 EndPoint:4	DATA1 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98
362	000000390	OUT	Add:2 EndPoint:5	DATA1 99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3 A4 A5 A6 A7 A8 A9
363	000000391	OUT	Add:2 EndPoint:6	DATA1 AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB
364	000000392	OUT	Add:2 EndPoint:7	DATA1 BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE
365	000000393	IN	Add:2 EndPoint:1	DATA1 A4 A3 A2 A1 A0 9F 9E 9D 9C 9B 9A 99 98 97 ACK
366	000000395	IN	Add:2 EndPoint:2	DATA1 96 95 94 93 92 91 90 8F 8E 8D 8C 8B 8A 89 88
367	000000396	IN	Add:2 EndPoint:3	DATA1 87 86 85 84 83 82 81 80 7F 7E 7D 7C 7B 7A 79 78
368	000000397	IN	Add:2 EndPoint:4	DATA1 77 76 75 74 73 72 71 70 6F 6E 6D 6C 6B 6A 69 68 67
369	000000398	IN	Add:2 EndPoint:5	DATA1 66 65 64 63 62 61 60 5F 5E 5D 5C 5B 5A 59 58 57 56
370	000000399	IN	Add:2 EndPoint:6	DATA1 54 53 52 51 50 4F 4E 4D 4C 4B 4A 49 48 47 46 45 44
371	000000400	IN	Add:2 EndPoint:7	DATA1 41 40 3F 3E 3D 3C 3B 3A 39 38 37 36 35 34 33 32
372	000000401	OUT	Add:2 EndPoint:1	DATA0 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2
373	000000402	OUT	Add:2 EndPoint:2	DATA0 E7 E8 E9 EA EB EC ED EE EF F0 F1 F2 F3 F4 F5 F6 F7

USBee RX User's Manual

Usbextractor -O output.dex -P 3209 -G -Q 10000 -R 1 -B

```
V - C:\Program Files\USBee AX-Pro Data Extractors\output.dex
File Edit View Favorites UserCommands GridLines Tools Help
[Icons]
0 10 20 30 40 50 60 70
123456789 123456789 123456789 123456789 123456789 123456789 123456789 123456789
1: 80 06 00 01 00 00 40 00
2: 12 01 00 01 FF FF FF 40 47 05 31 21 04 00 00 00 00 01
3:
4: 00 05 01 00 00 00 00 00
5:
6: 80 06 00 01 00 00 12 00
7: 12 01 00 01 FF FF FF 40 47 05 31 21 04 00 00 00 00 01
8:
9: 80 06 00 02 00 00 09 00
10:
11: 80 06 00 02 00 00 FF 00
12: 09 02 DA 00 01 01 00 80 32 09 04 00 00 00 FF FF FF 00 09 04 00 01 0D FF FF FF
13: 86 02 40 00 00 07 05 06 02 40 00 00 07 05 88 01 10 00 01 07 05 08 01 10 00 01
14: 05 81 03 40 00 0A 07 05 82 02 40 00 00 07 05 02 02 40 00 00 07 05 84 02 40 0C
15: 89 01 10 00 01 07 05 09 01 10 00 01 07 05 8A 01 10 00 01 07 05 0A 01 10 00 01
16:
17: 80 06 00 01 00 00 12 00
18: 12 01 00 01 FF FF FF 40 47 05 31 21 04 00 00 00 00 01
19:
20: 09 02 DA 00 01 01 00 80 32
21:
22: 80 06 00 02 00 00 EA 00
23: 09 02 DA 00 01 01 00 80 32 09 04 00 00 00 FF FF FF 00 09 04 00 01 0D FF FF FF
24: 86 02 40 00 00 07 05 06 02 40 00 00 07 05 88 01 10 00 01 07 05 08 01 10 00 01
25: 05 81 03 40 00 0A 07 05 82 02 40 00 00 07 05 02 02 40 00 00 07 05 84 02 40 0C
26: 89 01 10 00 01 07 05 09 01 10 00 01 07 05 8A 01 10 00 01 07 05 0A 01 10 00 01
27:
28: 00 09 01 00 00 00 00 00
29:
30: 01 0B 00 00 00 00 00 00
Lines 1 to 30 96% File Size: 1.73 KB (31 lines) [07/17/2006 16:35]
```

Usbextractor -O output.dex -P 3209 -G -Q 10000 -R 1 -A -H -V 1 -B -  
D

	0	10	20	30	40	50
1:	128 006 000 001 000 000 064 000					
2:	018 001 000 001 255 255 255 064	071 005 049 033 004 000				
3:						
4:	000 005 001 000 000 000 000 000					
5:						
6:	128 006 000 001 000 000 018 000					
7:	018 001 000 001 255 255 255 064	071 005 049 033 004 000				
8:						
9:	009 002 218 000 001 001 000 128 050					
10:						
11:	128 006 000 002 000 000 255 000					
12:	009 002 218 000 001 001 000 128	050 009 004 000 000 000				
13:	134 002 064 000 000 007 005 006	002 064 000 000 007 005				
14:	005 129 003 064 000 010 007 005	130 002 064 000 000 007				
15:	137 001 016 000 001 007 005 009	001 016 000 001 007 005				
16:						
17:	128 006 000 001 000 000 018 000					
18:						
19:	009 002 218 000 001 001 000 128 050					
20:						
21:	128 006 000 002 000 000 234 000					
22:	009 002 218 000 001 001 000 128	050 009 004 000 000 000				
23:	134 002 064 000 000 007 005 006	002 064 000 000 007 005				
24:	005 129 003 064 000 010 007 005	130 002 064 000 000 007				
25:	137 001 016 000 001 007 005 009	001 016 000 001 007 005				
26:						
27:	000 009 001 000 000 000 000 000					
28:						
29:	001 011 000 000 000 000 000 000					
30:						

# USBee RX User's Manual

V - C:\Program Files\USBee AX-Pro Data Extractors\output.dex

File Edit View Favorites UserCommands GridLines Tools Help

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF

```

00000000 12 01 00 01 FF FF FF 40 47 05 31 21 04 00 00 00 00  .yyy@G !
00000010 00 01 0A 0A 00 05 01 00 00 00 00 00 0A 0A 12 01  .
00000020 00 01 FF FF FF 40 47 05 31 21 04 00 00 00 00 01  .yyy@G !
00000030 0A 0A 80 06 00 02 00 00 09 00 0A 09 02 DA 00 01  . 12 . y
00000040 01 00 80 32 0A 0A 80 06 00 02 00 00 FF 00 0A 09  . 12 . yyy
00000050 02 DA 00 01 01 00 80 32 09 04 00 00 00 FF FF FF  . 12 . yyy
00000060 00 09 04 00 01 0D FF FF FF 07 05 81 03 10 00  . yyy
00000070 0A 07 05 82 02 40 00 00 07 05 02 02 40 00 00 07  . @ @ @
00000080 05 84 02 40 00 00 07 05 04 02 40 00 00 07 05 0A  . @ @ @
00000090 86 02 40 00 07 05 06 02 40 00 07 05 06 07 05 88 01  . @ @ @
000000A0 10 00 01 07 05 08 01 10 00 01 07 05 89 01 10 00  . |
000000B0 01 07 05 09 01 10 00 01 07 05 8A 01 10 00 01 07  . |
000000C0 05 0A 01 10 00 01 09 04 00 02 0D FF FF FF 00 07  . yyy
000000D0 0A 05 81 03 40 00 0A 07 05 82 02 40 00 00 07 05  . @ @ @
000000E0 02 02 40 00 00 07 05 84 02 40 00 00 07 05 04 02  . @ @ @
000000F0 40 00 00 07 05 86 02 40 00 07 05 06 02 40 00  . @ | @ @
00000100 00 07 05 88 01 00 01 01 07 05 08 01 00 01 01 07  . |
00000110 05 0A 89 01 10 00 01 07 05 09 01 10 00 01 07 05  . |
00000120 8A 01 10 00 01 07 05 0A 01 10 00 01 0A 0A 80 06  . |
00000130 00 01 00 00 12 0A 0A 12 01 00 01 FF FF FF 40 47  . yyy@G
00000140 05 31 21 04 00 00 00 00 01 0A 0A 80 06 00 02 00  . 1
00000150 00 09 00 0A 0A 80 06 00 02 00 00 EA 00 0A 80 02  . y y e
00000160 40 00 00 07 05 06 02 40 00 00 07 05 88 01 10 00  . @ @ |
00000170 01 07 05 08 01 10 00 01 07 05 89 01 10 00 01 07  . |
00000180 05 09 01 10 00 01 07 05 8A 01 10 00 01 07 05 0A  . |
00000190 01 10 00 01 09 04 00 02 0D FF FF FF 00 07 0A 05  . yyy
000001A0 81 03 40 00 0A 07 05 82 02 40 00 00 07 05 02 02  . @ @ @
000001B0 40 00 00 07 05 84 02 40 00 00 07 05 04 02 40 00  . @ | @ @
000001C0 00 07 05 86 02 40 00 00 07 05 06 02 40 00 00 07  . @
000001D0 05 88 01 00 01 01 07 05 08 01 00 01 01 07 05 0A  . |
000001E0 89 01 10 00 01 07 05 09 01 10 00 01 07 05 8A 01  . |
000001F0 10 00 01 07 05 0A 01 00 01 0A 0A 00 09 01 00 01  . |

```

Lines 1 to 32 96% File Size: 528 bytes (33 lines) [07/17/2006 16:32]

## CAN DATA EXTRACTOR

The CAN Bus Data Extractor takes the real-time streaming data from the CAN bus, formats it and allows you to save the data to disk or process it as it arrives.

### CAN DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- Monitors one CAN Bus
- intended to be used on the digital side of a CAN bus transceiver (such as the Microchip MCP2551)
- 11 or 29-bit identifier supported
- Time Stamp for each packet
- Output to Text File\*
- Output to Screen\*
- Comma or Space Delimited files
- Packet filter on Identifier
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

\* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

### HARDWARE SETUP

To use the Data Extractor you need to connect the USBee RX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

The CAN Bus Data Extractor connects to the digital side of your CAN bus transceiver and only needs to listen to the receiving side of the transceiver (such as the RxD pin on the Microchip MCP2551 CAN bus transceiver chip). Use signal 0 as the RxD data line and connect the GND line to the digital ground of your system. Connect these signals to the CAN bus transceiver IC using the test clips provided.

### EXTRACTOR COMMAND LINE PROGRAM

The CAN Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee RX software on your PC
- Plug in your USBee RX Test Pod into your PC using a USB 2.0 High Speed Port



- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- Run the executable using the following command line arguments:

```
CANExtractor [-?SDHICGAB] [-R CANSpeed] [-Q NumberOfBytes] [-V
Timestamp] [-O filename] [-M MaxID] [-N MinID] -P
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

R - Bus Speed in bits/second (default = 250000)

A - All Packet Fields are output (default)

B - Only Data Bytes are output

D - Decimal Text Values ("49")

H - Hex Text Values ("31") default

I - Binary Values (49)

C - Comma Delimited

G - Space Delimited (default)

M - Maximum Identifier Filter

N - Minimum Identifier Filter

V - Timestamps (0=off, 1=each packet start)

## EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee RX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

---

## DLL FILENAME:

usbedCAN.dll in \Windows\System32

---

## DLL EXPORTED FUNCTIONS AND PARAMETERS

**ExtractionBufferCount** – Returns the number of bytes that have been extracted from the data stream so far and are available to read using `GetNextData`.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

**GetNextData** – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,  
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

**StartExtraction** – Starts the Data Extraction with the given parameters.

```
CWAV_IMPORT int CWAV_API StartExtraction(unsigned long Threshold,  
unsigned long PodNumber, unsigned long Speed, unsigned char All,  
unsigned char Decimal, unsigned char Hex, unsigned char Binary,  
unsigned char Comma, unsigned char Space, unsigned char Timestamps,  
unsigned long MaxID, unsigned long MinID);
```

Threshold: The voltage above which is a logic 1 and below which is a logic 0 (times 10). 10 = 1.0V, 15 = 1.5V

PodNumber: Pod ID on the back of the USBee RX Test Pod

Speed: Bit rate of the CAN bus in bits per second

All:

- 0 – Only the data payload bytes are returned
- 1 – All CAN packet fields are returned

Decimal:

- 1 – Decimal Values (text) are output for the data bytes

Hex:

- 1 – Hex Values (text) are output for the data bytes

Binary:

- 1 – All data is in binary form, not text

Comma:

- 1 – Commas are placed between each field/data byte

Space:

- 1 – Spaces are placed between each field/data byte

Timestamp:

- 1 – Print Timestamps at the start of each packet

MaxIDFilter:

- The Maximum Identifier to log (0xFFFFFFFF default)

MinIDFilter:

- The Minimum Identifier to log (0 default)

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

**StopExtraction** – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

**ExtractBufferOverflow** – Returns the state of the overflow conditions

```
CWAV__EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

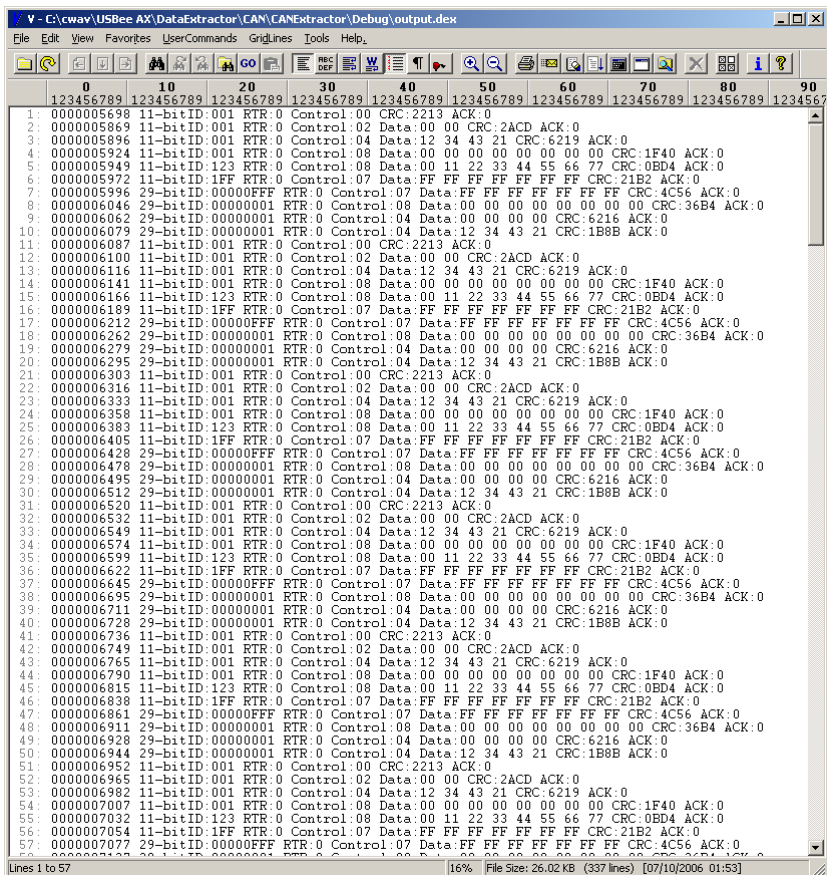
- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

## EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the \*buffer parameter.

The CAN Bus Extraction DLL sends the extracted data through the \*buffer in the requested form based on the parameters in the StartExtraction call. For example, if Binary is set to a 0, then the \*buffer will receive the binary bytes that make up the data stream. If Hex is set to a 1, the \*buffer will contain a text string which is the data of the CAN traffic in Hex text form, separated by any specified delimiters.

```
CANExtractor -O output.dex -S -P 3209 -Q 500000 -R 250000 -A -H -V 1
```



```
1: 0000005698 11-bitID:001 RTR:0 Control:00 CRC:2213 ACK:0
2: 0000005869 11-bitID:001 RTR:0 Control:02 Data:00 00 CRC:2ACD ACK:0
3: 0000005896 11-bitID:001 RTR:0 Control:04 Data:12 34 43 21 CRC:6219 ACK:0
4: 0000005924 11-bitID:001 RTR:0 Control:08 Data:00 00 00 00 00 00 CRC:1F40 ACK:0
5: 0000005949 11-bitID:123 RTR:0 Control:08 Data:00 11 22 33 44 55 66 77 CRC:0BD4 ACK:0
6: 0000005972 11-bitID:1FF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:21B2 ACK:0
7: 0000005996 29-bitID:0000FFFF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:4C56 ACK:0
8: 0000006046 29-bitID:00000001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 CRC:36B4 ACK:0
9: 0000006062 29-bitID:00000001 RTR:0 Control:07 Data:00 00 00 00 00 00 CRC:6216 ACK:0
10: 0000006079 29-bitID:00000001 RTR:0 Control:04 Data:12 34 43 21 CRC:1B8B ACK:0
11: 0000006087 11-bitID:001 RTR:0 Control:00 CRC:2213 ACK:0
12: 0000006100 11-bitID:001 RTR:0 Control:02 Data:00 00 CRC:2ACD ACK:0
13: 0000006116 11-bitID:001 RTR:0 Control:04 Data:12 34 43 21 CRC:6219 ACK:0
14: 0000006141 11-bitID:001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 CRC:1F40 ACK:0
15: 0000006166 11-bitID:123 RTR:0 Control:08 Data:00 11 22 33 44 55 66 77 CRC:0BD4 ACK:0
16: 0000006189 11-bitID:1FF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:21B2 ACK:0
17: 0000006212 29-bitID:0000FFFF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:4C56 ACK:0
18: 0000006262 29-bitID:00000001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 CRC:36B4 ACK:0
19: 0000006279 29-bitID:00000001 RTR:0 Control:04 Data:00 00 00 00 00 00 CRC:6216 ACK:0
20: 0000006295 29-bitID:00000001 RTR:0 Control:04 Data:12 34 43 21 CRC:1B8B ACK:0
21: 0000006303 11-bitID:001 RTR:0 Control:00 CRC:2213 ACK:0
22: 0000006316 11-bitID:001 RTR:0 Control:02 Data:00 00 CRC:2ACD ACK:0
23: 0000006333 11-bitID:001 RTR:0 Control:04 Data:12 34 43 21 CRC:6219 ACK:0
24: 0000006358 11-bitID:001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 CRC:1F40 ACK:0
25: 0000006383 11-bitID:123 RTR:0 Control:08 Data:00 11 22 33 44 55 66 77 CRC:0BD4 ACK:0
26: 0000006405 11-bitID:1FF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:21B2 ACK:0
27: 0000006428 29-bitID:0000FFFF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:4C56 ACK:0
28: 0000006478 29-bitID:00000001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 CRC:36B4 ACK:0
29: 0000006495 29-bitID:00000001 RTR:0 Control:04 Data:00 00 00 00 00 00 CRC:6216 ACK:0
30: 0000006512 29-bitID:00000001 RTR:0 Control:04 Data:12 34 43 21 CRC:1B8B ACK:0
31: 0000006520 11-bitID:001 RTR:0 Control:00 CRC:2213 ACK:0
32: 0000006532 11-bitID:001 RTR:0 Control:02 Data:00 00 CRC:2ACD ACK:0
33: 0000006549 11-bitID:001 RTR:0 Control:04 Data:12 34 43 21 CRC:6219 ACK:0
34: 0000006574 11-bitID:001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 CRC:1F40 ACK:0
35: 0000006599 11-bitID:123 RTR:0 Control:08 Data:00 11 22 33 44 55 66 77 CRC:0BD4 ACK:0
36: 0000006622 11-bitID:1FF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:21B2 ACK:0
37: 0000006645 29-bitID:0000FFFF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:4C56 ACK:0
38: 0000006695 29-bitID:00000001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 CRC:36B4 ACK:0
39: 0000006711 29-bitID:00000001 RTR:0 Control:04 Data:00 00 00 00 00 00 CRC:6216 ACK:0
40: 0000006728 29-bitID:00000001 RTR:0 Control:04 Data:12 34 43 21 CRC:1B8B ACK:0
41: 0000006736 11-bitID:001 RTR:0 Control:00 CRC:2213 ACK:0
42: 0000006749 11-bitID:001 RTR:0 Control:02 Data:00 00 CRC:2ACD ACK:0
43: 0000006765 11-bitID:001 RTR:0 Control:04 Data:12 34 43 21 CRC:6219 ACK:0
44: 0000006790 11-bitID:001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 CRC:1F40 ACK:0
45: 0000006815 11-bitID:123 RTR:0 Control:08 Data:00 11 22 33 44 55 66 77 CRC:0BD4 ACK:0
46: 0000006838 11-bitID:1FF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:21B2 ACK:0
47: 0000006861 29-bitID:0000FFFF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:4C56 ACK:0
48: 0000006911 29-bitID:00000001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 CRC:36B4 ACK:0
49: 0000006928 29-bitID:00000001 RTR:0 Control:04 Data:00 00 00 00 00 00 CRC:6216 ACK:0
50: 0000006944 29-bitID:00000001 RTR:0 Control:04 Data:12 34 43 21 CRC:1B8B ACK:0
51: 0000006952 11-bitID:001 RTR:0 Control:00 CRC:2213 ACK:0
52: 0000006965 11-bitID:001 RTR:0 Control:02 Data:00 00 CRC:2ACD ACK:0
53: 0000006982 11-bitID:001 RTR:0 Control:04 Data:12 34 43 21 CRC:6219 ACK:0
54: 0000007007 11-bitID:001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 CRC:1F40 ACK:0
55: 0000007032 11-bitID:123 RTR:0 Control:08 Data:00 11 22 33 44 55 66 77 CRC:0BD4 ACK:0
56: 0000007054 11-bitID:1FF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:21B2 ACK:0
57: 0000007077 29-bitID:0000FFFF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:4C56 ACK:0
```

## GETTING HELP

We are always eager to help you to get the most out of all USBee products. If you have any questions, comments, bug reports or suggestions, please contact us. We actively improve our product line and your feedback is the key to making the USBee the best embedded development tool on the market.

Email us at [support@usbee.com](mailto:support@usbee.com)

or

Call us at (951) 694-6808

Copyright 2012 CWAV. All Rights Reserved

Printed in the USA

Version 1.5

USBee RX User's Manual

267