

Technical information note 9:

Capturing and processing RDT3000 data on the Etherlog

Document revision history		
Created	28 Apr 2004	Mike Lowndes
Updated command structure	27 July 2005	Mike Lowndes

1 Introduction

The low cost RDT 3000 telemetry module can be used to expand the input capability of the Etherlog 3000. The RDT 3000 measures eight analogue inputs and counts pulses on four digital inputs, and periodically broadcasts radio packets containing this information, along with the instantaneous digital input states, using its built-in 433MHz licence-exempt radio transmitter. The Etherlog 3000 can be configured to listen for these broadcasts and save the data from up to eight RDT 3000s, expanding the effective number of analogue channels by 64 and digital inputs/pulse counters by 32.

Etherlog 3000 logging tasks can access and process the data from the RDTs, mixing it as required with data measured on the Etherlog. The data represents the last-received data from each RDT, each new receipt overwriting the previous one.

This Technical Information Note describes in detail how to set up the Etherlog and RDTs to achieve this.

2 RDT 3000 fundamentals

The RDT 3000 is programmed using a rotary switch on its circuit board, which is accessible with the lid off. The switch has sixteen positions which determine the period between broadcasts, which varies from 10 seconds to 20 minutes, and whether or not the broadcasts are encrypted. Each broadcast contains the current values of all eight analogue inputs, all four pulse counters and the digital input states, so the RDT does not have to be configured with information about what to measure and transmit. More importantly, this makes it possible for the Etherlog to know that each entry in a broadcast corresponds to a particular analogue or digital channel, so system configuration information is only needed on the Etherlog.

Since radio is a shared medium, it is possible for interference to corrupt transmissions. All broadcasts contain CRC error check codes which are inspected on receipt on the Etherlog. Corrupt packets are automatically discarded, resulting in the loss of data. In order to avoid this as far as possible, the RDT rebroadcasts the same data after a randomised delay of a few seconds, so that if the original was lost, the Etherlog may be able to receive and use the repeated one. If the Etherlog receives both, it simply discards the repeat.

3 Etherlog 3000 configuration

The Etherlog 3000 must be set up to listen for RDT 3000 broadcasts, which by default it does not do. Each RDT has a unique identifier (or “address”) in the range 1 to 65535, and this is attached to each broadcast. The address is used by the Etherlog to decide whether to store the information in the packet, or whether to ignore it.

Since quoting each RDT’s unique identifier in *lucid* programs would be an inconvenience, particularly when otherwise identical programs are to be used in several installations, RDTs are defined also by an “index” into an internal table which stores their details.

Using the command line interface, or entries in `startup.cmd`, an RDT whose address is 35 is added to the first entry in the table of RDTs the Etherlog listens for using the following command:

```
rdt add 1 35
```

It is possible to configure up to eight RDTs using this command, using the numbers 1 .. 8 to indicate their indices. Once RDTs are defined in this way, all other operations (either via command line or from *lucid* programs) quote the index, not the address.

The above-defined RDT can be removed from the list using the command:

```
rdt delete 1
```

You can find which RDTs are currently in the list by issuing the command:

```
rdt status
```

which prints a list of configured RDT indices and addresses, their transmission periods, their decryption keys (which will be explained in a later section), and time of receipt of last broadcast. An example status report is shown below, illustrating two configured RDTs in index locations 3 and 7:

Idx	Addr	Per	Key	Last
3	35	10	00000000	14:27:33 13/04/2004
7	42	30	02F60CFD	14:27:39 13/04/2004

When there are any RDTs configured, the Etherlog cannot sleep, since it could miss broadcasts. Thus sleeping is disabled as soon as the first RDT is added, and re-enabled when the last one is deleted. Using the battery as the primary source of power may not be a good thing to do when RDTs are enabled – it may be advisable to use mains or solar power in this case. However, use of *lucid* commands to disable and enable the RDT system (thereby enabling and disabling sleeping) may make it possible under some circumstances to use battery power with RDTs.

When the power is removed from the Etherlog, it loses the RDT configuration. To ensure that the required RDTs are enabled when the system powers up again, put the `rdt add` commands into `startup.cmd`.

4 Accessing RDT data from the command line

Before using the command line to interrogate RDTs, it is important to realise that if the connection to the logger is via the low power radio link, RDT transmissions will not be received. Thus the status and input values displayed will be the last ones received before the radio connection was established. If a new RDT is commissioned while a radio link is active, it will be necessary briefly to disconnect before its operation can be confirmed.

Three further command line interface instructions can be issued, which enable you to check the latest values of analogue, digital and pulse channels in a manner similar to the built-in Etherlog channels. These are:

```
rdt ain <index> <channel>
rdt din <index> <channel>
rdt pulse <index> <channel>
```

In each case, `<index>` is the index of the RDT you want to check, and `<channel>` is the channel number of the input. For example,

```
rdt ain 7 3
```

reports the value, in microvolts, of analogue channel 3 on RDT whose index is 7. Recall from the previous section that the RDT with address 42 has been registered with this index. Channel numbers range from 1 to 8 for analogue inputs, and from 1 to 4 for digital and pulse channels.

5 Accessing RDT data from *lucid* tasks

The *lucid* programming language contains commands which allow you to access RDT data in a similar manner to the built-in analogue, digital and pulse channels. These are:

```
rdtain <index> <channel>
rdtdin <index> <channel>
rdtpulse <index> <channel>
```

Note the lack of a space between the “rdt” and “ain” etc. – each is a single word.

In each case, the <index> and <channel> values are the same as in the command line versions. The returned values are in integer microvolts for analogue channels, 1 or 0 for digital inputs, and integer for pulse counts.

Note that the range of values returned by `rdtain` is from 0 to 2499390, since the voltage reference on the RDT is 2.5V instead of the 512mV reference used on the Etherlog, and RDT inputs are converted by a unipolar 12-bit analogue-digital converter (thus $4095/4096 \times 2.5V$ is 2.499390V).

While this is not a problem if you are using the RDT to measure an absolute voltage, you will get an erratic reading when measuring temperature using `fcmt2c`, `fpt2c` or `fpt1k2c` commands with the `rdtain` result as its argument. The three temperature commands assume that a voltage divider has been set up with the appropriate sensor, and that the reference voltage is 512mV. Thus the voltage readings from the `rdtain` command need to be scaled down by a factor of 0.512/2.5 before being fed into `fcmt2c` etc. The following line of *lucid* does this:

```
fcmt2c fmul 0.2048 rdtain 7 1
```

This reads RDT index 7 analogue channel 1, multiplies it by the required scale factor, and then converts it to the equivalent temperature.

A further *lucid* command, `rdttime <index>`, accesses the timestamp of the last broadcast received from a particular RDT. This enables you to check that data from the RDT is reasonably recent – you may wish to ignore it if broadcasts from the RDT have not reached the Etherlog for more than an hour, say. This command takes one argument, the address of the RDT. The example below reads and records analogue input 1 of RDT 35 if data has been received in the last 30 minutes, else records the value 0.

```
if < sub time rdttime 7 1800
  write rdtain 7 1
else
  write 0
endif
```

6 Selecting the RDT broadcast period

The RDT broadcast period that you use will be influenced by (i) how “out-of-date” you can allow RDT data to be in your application, and (ii) how often RDT broadcasts are missed by the Etherlog.

Ideally, you would like RDT data to be as up-to-date as possible, but real system constraints, such as the expected battery life of the RDT at various broadcast intervals, will generally dictate that the RDT does not broadcast as often as the 10 second minimum interval. Where the RDT is measuring data such as ambient temperature that does not vary rapidly, and that is being logged by the Etherlog say each 15 minutes, it is probably reasonable to set the RDT to broadcast every 5 minutes, which would give a typical and worst-case data age of 2.5 and 5 minutes respectively (in the absence of lost transmissions).

Where broadcasts are being lost (through intermittent interruption of the signal path, or through occasional clashes with other RDTs), it may be necessary to reduce the broadcast interval in an attempt to improve the availability of recent data. Bear in mind, though, that increased radio activity may in itself lead to increased collision rates.

7 Enabling and disabling RDT reception in *lucid*

There may be some circumstances where you do not need RDT data to be received all of the time, and wish to control reception from within a *lucid* task. The command `rdtenable <flag>` permits this, where the value of `flag` is 0 to disable RDT reception, and non-zero to enable it. Disabling RDT reception in this way does not affect the RDT setup as defined by the `rdt add` commands, but does allow the Etherlog to sleep, saving battery power.

It is possible to set up a *lucid* task to enable RDT reception some time before RDT data is to be recorded, allowing enough time for RDT data to be received, then record the data and disable RDT reception again.

In the following simple example, RDT data is to be recorded every hour from two RDTs whose broadcast interval is 2 minutes. At 5 minutes before the next hourly recording we turn enable RDT reception. Five minutes later when the task next runs, we detect that we are now on or just past the hour, so we record the data and turn RDT reception off again. This means that the Etherlog can sleep for 55 minutes in every hour between 5 minute task activations, staying awake only when RDT reception is enabled.

```
repeat 300                # run task every 5 minutes

begintask

    # if now 5 minutes to the hour, enable RDT reception
    if >= mod time 3600 3300
        rdtenable 1
    endif

    # if now on or just past the hour, record data
    if < mod time 3600 150
        newline
        write time
        write rdtain 3 1
        write rdtain 7 1
        write rdtpulse 7 1
        rdtenable 0        # disable RDT reception
    endif

endtask
```

8 Pulse counter data

The pulse counter channels in the RDT are implemented as 16 bit counters, but the Etherlog extends these to 32 bit counters using overflow detection. In *extreme* conditions, though, it is possible to miss an overflow, thus losing 65536 pulses.

As an example, consider an RDT counting a 10Hz pulse train, and configured to broadcast at 20 minute intervals. The counter values transmitted by the RDT will be as follows:

1st	12000
2nd	24000
3rd	36000
4th	48000
5th	60000
6th	6464
7th	18464

The sixth broadcast contains a count of 6464 because the RDT's counter overflowed at 65536 and returned to zero. The seventh contains 18464, which is 12000 + 6464.

Suppose the first broadcast is correctly received by the Etherlog. When it receives the 6th one, it sees that 6464 is less than the previous reading, and compensates for this by adding 65536 to the 32 bit internal value. If, however, it does not receive the 2nd, 3rd, 4th, 5th or 6th broadcasts, it cannot spot the overflow because 18464 in the 7th broadcast is greater than the previous received count of 12000, and 65536 pulses have been lost.

It should be noted that this really is an extreme condition which is unlikely to occur in real applications where the Etherlog and RDT are in reasonable range. If you really want to count pulses at this frequency, it would be a good idea to use a shorter transmission period of 2 or 5 minutes so that many more consecutive broadcast losses would have to occur before the overflow detection failure occurs.

9 RDT Encryption

In order to protect data from unwanted "snooping", RDTs can be configured to transmit their data in encrypted form. Eight of the sixteen switch positions on the RDT duplicate the selectable transmission intervals but turn on the encryption feature.

Encryption is performed only on the data part of each broadcast. The encryption algorithm uses a 32 bit "key" unique to each RDT: this is hardwired into the RDT and must be known by the Etherlog in order to decrypt the broadcast. The Etherlog can tell from the control part of the broadcast whether it is encrypted, and automatically decrypts it before storing the data.

The key for a particular RDT is configured from the command line using the `rdt add` command, but with an extra argument which gives the key in hexadecimal form (as given in the documentation supplied with the RDT), as this example shows:

```
rdt add 1 35 02f60cfd
```

The key is shown in the `rdt status` display described above, and enables you to check that the correct value is being used.