

## Kelunji Telemetry Data Formats

Greg McPherson December 20, 1995.

### Type 1: Original 14 bit Data

Data is read from the Kelunji as 8 bit bytes.

The first bit of each byte determines whether the byte is a high or low byte.

The first bit of the high byte must be a "0" and the first bit of the low byte must be a "1" or an error has occurred.

The remaining 7 bits of each byte have relevant data,

ie. in 0111 1111 the "1's" are useful data.

A high byte then a low byte is sent and should be converted into a 16 bit word as follows:

The high byte is shifted to the left 7 bits, then the 7 lower bits of the low byte are added to this.

ie. if the high byte is "0AAA AAAA", and the low byte is "1BBB BBBB" they are combined to form the word "00AA AAAA ABBB BBBB"

Finally if the 3rd highest bit is a "1", ie "001A AAAA ABBB BBBB", then the highest two bits are also set to be "1's", giving

"111A AAAA ABBB BBBB". This is a negative number using the "Two's Complement" method of expressing negatives.

In Decimal, data values for Type 1 range from -8192 to +8191.

### Type 2: 13 bit Data plus Status

Data is read from the Kelunji as 8 bit bytes with 7 bits of relevant data, with a "0" in the first bit representing a high byte and a "1" representing a low byte, the same as for Type 1.

Every 2 bytes is combined into a 16 bit word as follows:

The 7th bit of the high byte is examined. If it is a "1" then we have a **data** sample, if it is a "0" then we have a **status** sample.

ie. "01AA AAAA" is the high byte of a **data** sample.

"00AA AAAA" is the high byte of a **status** sample.

If we have a **data** sample, we combine the 2 bytes in the similar manner to Type 1, giving a 16 bit word as follows:

The first byte is shifted to the left 7 bits, then the 7 lower bits of the second byte are added to this.

ie. if the high byte is "01AA AAAA", and the low byte is "1BBB BBBB" they are combined to form the word "001A AAAA ABBB BBBB"

Finally if the **4th** highest bit is a "1", ie "001**1** AAAA ABBB BBBB", then the highest two bits are set to be "1's", giving "1111 AAAA ABBB BBBB". This is a negative number using the "Two's Complement" method of expressing negatives.

In Decimal, data values for Type 2 data range from -4096 to +4095.

If we have a **status** sample, we extract two numbers from the two bytes. We extract a **value** and a **code**.

**Decoding the code:**

Shift the high byte 3 bits to right then "and" it with "0000 0111".  
ie. if we have the byte "00AB CDEF", we get "0000 0ABC",  
which is a code number between 0 and 7.

Each code represents a different piece of status information:

code = 0 :new second  
code = 1 :Battery voltage  
code = 2 :Supply Current  
code = 3 :Charger Current  
code = 4 :Number of Events  
code = 5 :Storage Details  
code = 6 :Temperature  
code = 7 :Status bits

**Decoding the value:**

We "and" the high byte with "0000 0111" then shift it left 7 bits, then add the 7 lower bits of the low byte.

ie. if the first byte is "00AB CDEF" and the second byte is "1abc defg"  
we get "00DE Fabc defg", which is a number between 0 and 1023.

Depending on the code, the value is interpreted in different ways:

**If code = 0**

We are receiving data on the date or time.

We have to re-examine the two bytes to find another code which states which part of the date/time is being sent, and another value.

**The new code:** We "and" the high byte with "0000 0111".

ie. if the first byte is "00AB CDEF", we get "0000 0DEF", which is a number between 0 and 7.

**The new value:** We use the value of the 7 lower bits of the low byte, ie. "0abc def", which is a number between 0 and 127.

if the new code = 0, then the new value = the current second.

if the new code = 1, then the new value = a new minute,  
that is, the minute = new value, and the seconds should be set to zero.

if the new code = 2, then the new value = a new hour.

" " " 3, " " " a new day.

" " " 4, " " " a new month.

" " " 5, " " " a new year.

**If code = 1**

We are receiving data on the Battery Voltage.

Voltage = value x 0.02

**If code = 2**

We are receiving data on the Supply Current.

Current = value

**If code = 3**

We are receiving data on the Charger Current

Current = value x 8

**If code = 4**

We are receiving data on the number of triggers recorded by this Kelunji.

Triggers = value

**If code = 5**

We are receiving Storage details.

Percent of free memory = low seven bits of value.

ie. If the value is (in binary) "00DE Fabc defg"

then the percent of free memory = "0000 0abc defg".

Megabytes installed in the Kelunji = value shifted right 7 bits.

ie. If the value is (in binary) "00DE Fabc defg" then

Megabytes = "0000 0000 0DEF", which is a number between 0 and 7.

**If code = 6**

We are receiving the temperature of the Kelunji.

Temperature = value - 50

**code = 7**

is currently not used.

## Code from SeisMonitor for Decoding Kelunji Data

```
//                               DecodeSamples
//
#pragma segment MAEvtHandlerRes
u_int16
DecodeSamples (TChannel *theChannel, u_int8 buffer[], const int16
numBytes)
{
register int index;
u_int8      code;
int16      value;

theChannel->fELS = 0;
switch (theChannel->fChannelPref.dataFormat)
{
case1: // Original 14 bit data
for (index=0; index<numBytes; index++)
{
if ((buffer[index]&0x80) || (!(buffer[index+1]&0x80)))
{
theChannel->fELS++; // Count the error
}
else
{
value = (buffer[index]<<7) + (buffer[index+1]&0x7F);
if (value & 0x2000) value |= 0xC000; // Decode the
value

CheckAndStore (theChannel, value);
index++;
}
}
break;

case2: // 13 bit data plus status
for (index=0; index<numBytes; index++)
{
if ((buffer[index]&0x80) || (!(buffer[index+1]&0x80)))
{
theChannel->fELS++; // Count the error
}
else
{
if (buffer[index] & 0x40)
{
// Normal data sample
value = ((buffer[index]&0x3F)<<7) +
(buffer[index+1]&0x7F);
if (value & 0x1000) value |= 0xE000;
CheckAndStore (theChannel, value);
}
else
{
// Special status value
value = ((buffer[index]&0x07)<<7) +
(buffer[index+1]&0x7F);
code= (buffer[index]>>3) & 0x07;
switch (code)
{
case0: // New second
theChannel->fRecorderSample = theChannel-
>fSLS;

value = buffer[index+1] & 0x7F;
code= buffer[index] & 0x07;
switch (code)
```

```

    {
case0:                                // Just second
    theChannel->fRecorderSec = value;
    break;
case1:                                // New minute
    theChannel->fRecorderSec = 0;
    theChannel->fRecorderMin = value;
    break;
case2:                                // New hour
    theChannel->fRecorderSec = 0;
    theChannel->fRecorderMin = 0;
    theChannel->fRecorderHour = value;
    break;
case3:                                // New day
    theChannel->fRecorderSec = 0;
    theChannel->fRecorderMin = 0;
    theChannel->fRecorderHour = 0;
    theChannel->fRecorderDay = value;
    break;
case4:                                // New month
    theChannel->fRecorderSec = 0;
    theChannel->fRecorderMin = 0;
    theChannel->fRecorderHour = 0;
    theChannel->fRecorderDay = 1;
    theChannel->fRecorderMonth = value;
    break;
case5:                                // New year
    theChannel->fRecorderSec = 0;
    theChannel->fRecorderMin = 0;
    theChannel->fRecorderHour = 0;
    theChannel->fRecorderDay = 1;
    theChannel->fRecorderMonth = 1;
    theChannel->fRecorderYear = value;
    break;
    }
break;
case1:                                // Battery voltage
    theChannel->fBatteryVoltage=
        (extended)value*0.02;
    break;

case2:                                // Supply current
    theChannel->fSupplyCurrent = value;
    theChannel->fSumSupply += value;
    theChannel->fNumSupply++;
    break;

case3:                                // Charger current
    theChannel->fChargeCurrent = 8*value;
    theChannel->fSumCharge += 8*value;
    theChannel->fNumCharge++;
    break;

case4:                                // Number of events
    theChannel->fNumberOfTriggers = value;
    break;

case5:                                // Storage details
    theChannel->fPercentFree = value&0x7F;
    theChannel->fMegaBytes = value>>7;
    break;

case6:                                // The temperature

```

```
        theChannel->fTemperature = value - 50;
        break;

        case7:                // Status bits
            break;
        }
        value = theChannel->fLast1Sample +
(int16)theChannel->fDC;
        CheckAndStore (theChannel, value);
    }
    index++;
}
}
break;
}
return index;
}
```