Hit Message Feature IDs (values are decimal), data type and scaling:

Note that in the example code, p is a pointer to the first byte of the parameter in the AE hit data message (ID 1).

RISETIME (ID 1)
This is a 2byte value stored as an unsigned short in units of microsec

```
 unsigned short Risetime(const unsigned char *p)
{
   return(*(unsigned short *)p);
}
```

COUNTS TO PEAK (ID 2)
This is a 2byte value stored as an unsigned short, no units

```
unsigned short CountsPeak(const unsigned char *p)
{
   return(*(unsigned short *)p);
}
```

COUNTS (ID 3)
This is a 2byte value stored as an unsigned short, no units

```
unsigned short Counts(const unsigned char *p)
{
   return(*(unsigned short *)p);
}
```

ENERGY (ID 4)
This is a 2byte value stored as an unsigned short, no units

```
unsigned short Energy(const unsigned char *p)
{
   return(*(unsigned short *)p);
}
```

DURATION (ID 5)
This is a 4byte value stored as an unsigned long in units of microsec

```
unsigned long Duration(const unsigned char *p)
{
```

```
   return(*(unsigned long *)p);
}
```

AMPLITUDE (ID 6)
This is a 1byte value stored as an unsigned char in units of dB

```
unsigned char Amplitude(const unsigned char *p)
{
   return(*p);
}
```

RMS (ID 7) – 8 bit, obsolete
This is a 1byte value stored as an unsigned char in units of volts

```
float RMS8(const unsigned char *p)
{
   return(((float)(*p)) / 20.0f);          // scale to <=7.07V
}
```

ASL (ID 8)
This is a 1byte value stored as an unsigned char in units of dB

```
unsigned char Asl(const unsigned char *p)
{
   return(*p);
}
```

THRESHOLD (ID 10)
This is a 1byte value stored as an unsigned char in units of dB

```
unsigned char Threshold(const unsigned char *p)
{
   return(*p);
}
```

AVERAGE FREQUENCY (ID 13)
This is a 2byte value stored as an unsigned short in units of kHz

```
unsigned short AvgFreq(const unsigned char *p)
{
   return(*(unsigned short *)p);
}
```

RMS16 (ID 17)
Replaces the 8 bit RMS that was used in older systems.
This is a 2byte value stored as an unsigned short.
To scale to a float, divide the unsigned short value by 5000.

```
float RMS16(const unsigned char *p)
{
   return((float)(*(unsigned short *)p) / 5000.0f);
}
```

REVERBERATION FREQUENCY (ID 18)
This is a 2byte value stored as an unsigned short in units of kHz

```
unsigned short RevFreq(const unsigned char *p)
{
   return(*(unsigned short *)p);
}
```

INITIATION FREQUENCY ((ID 19)
This is a 2byte value stored as an unsigned short in units of kHz

```
unsigned short InitFreq(const unsigned char *p)
{
   return(*(unsigned short *)p);
}
```

SIGNAL STRENGTH (ID 20)
This is a 4 byte value stored as an unsigned long.
To scale it, multiply the stored value by 3.05

```
float SigStrength(const unsigned char *p)
{
   // data in message is in A/D units, we must convert to pVs
   return( (float)(*(unsigned long *)(p)) * 3.05f);
}
```

ABSOLUTE ENERGY (ID 21)
This is a 4 byte value stored as a float.
To scale it, multiply the stored value by 931e-6

```
float AbsEnergy(const unsigned char *p)
{
```

```
   // data in message is in A/D units, we must convert to
   // milli-attoJoules
   return(*(float *)p * 0.000931f);
}
```

PARTIAL POWER (ID 22)
This indicates the presence of one or more partial powers in the data set.
In order to know which ones, message 109 must be processed.
More on that later.

FREQUENCY CENTROID (ID 23)
This is a 2byte value stored as an unsigned short in units of kHz

```
unsigned short CentFreq(const unsigned char *p)
{
   return(*(unsigned short *)p);
}
```

FREQUENCY PEAK (ID 24)
This is a 2byte value stored as an unsigned short in units of kHz

```
unsigned short PeakFreq(const unsigned char *p)
{
   return(*(unsigned short *)p);
}
```

Note that the order in which features are defined in message 5 is the same order that they occur in the hit data message.

When partial powers are enabled, message 109 determines which and how many segments are present in the data set.

So if you use message 5 to determine the offsets of each feature in the hit data message, you will have to re-adjust the offsets for all features found after CHID 22 is found. The amount of the adjustment is one byte for each partial power segment defined.

Partial Power Setup Message (ID=109)

2 bytes	unsigned short	message length
1 byte	unsigned char	message id (=109 =0x6D)

1 byte	unsigned char	segment type (always 0)
2 bytes	unsigned short	number of segments defined in the message (if 0 then no partial powers are defined)
2 bytes	unsigned short	reserved, always = 1
2 bytes	unsigned short	reserved, always = 1
2 bytes	unsigned short	segment number (0 to 3, 4 segments maximum)
2 bytes	unsigned short	segment start (not required for processing the hit message)
2 bytes	unsigned short	segment end (not required for processing the hit message)

Example:

Message ID 5 is used to determine which AE features are present on the hit data set (message ID 1).
The features in message 1 appear in the same order as they are defined in message 5.

Example bytes from message 5:
0B 00 05 08 01 03 04 05 06 15 16 17 01

The first 2 bytes are the message length (little-endian LSB,MSB) so the length of the message is 11.
The next byte is the message ID (5=hit definition)
The next 10 (Length-1) are the body.
There are 8 AE feature ids. Using the ID values from above (converted to decimal):
1=risetime
3=counts
4=energy
5=duration
6=amplitude
21=absolute energy
22=partial powers
23=frequency centroid
The last byte is the number of parametric values (1) in the hit message (note this is not the parametric id).

Because partial powers are defined, message 109 also needs to be processed.
24 00 6D 00 04 00 00 00 00 00 03 00 01 00 04 00 09 00 02 00
0A 00 0D 00 03 00 0E 00 13 00 01 00 01 00 00 00 13 00

message length=36 (24 00)
message id=109 (6D)

segment type (always 0) (00)
number of segments defined in the message=4 (04 00)
For processing the hit message, the segment information below is not required, all that is important is how many segments there are.
segment number=0 (00 00)
segment start = 0 (00 00)
segment end = 3 (03 00)
segment number=1 (01 00)
segment start = 4 (04 00)
segment end = 9 (09 00)
segment number=2 (02 00)
segment start = 10 (0A 00)
segment end = 13 (0D 00)
segment number=3 (03 00)
segment start = 4 (0E 00)
segment end = 19 (13 00)
total power number of segments=1 (01 00)
total power segment number=1 (01 00)
total power segment start = 0 (00 00)
total power segment end = 19 (13 00)

```
Hit message in line display:
1   0 00:00:04.8508117     0.0095    98    16    31       108  70  46.878E+03
0       0       1      98   192
```

The bytes for a hit message:

```
20 00 01 EF 11 28 01 00 00 01 62 00 10 00 1F 00 6C 00 00 00
46 1C 14 40 4C 00 00 01 62 C0 00 01 1F 00
```

The first 2 bytes are the message length (little-endian LSB,MSB) so the length of the message is 32.
The next byte is the message ID (1=AE hit)
The next 31 (Length-1) are the body.
The first AE data is the time of test. It uses 6 bytes (EF 11 28 01 00 00)
The next is the AE channel ID (01) so this hit is from channel 1.
The next are the AE features as defined by message 5.
Risetime=98 (62 00)
Counts=16 (10 00)
Energy=31 (1F 00)
Duration=108 (6C 00 00 00)
Amplitude=70 (46)
absolute energy= 46.878E+03 (1C 14 40 4C) = 5.035224E7*0.000931

partial powers= 0,0,1,98 (00 00 01 62)
frequency centroid=192 (C0 00)
parametric 1= 0.0095 (1F 00) = 31 * 10 / 32768