FAZT I4 Data Transmission Format

Author:
Faz Technology

Contents

1 Document History

Issue	Date	Authors	Comment
1.0	19 May 2016	Faz Technology	Initial release
1.1	25 July 2016	Faz Technology	Added section for Timestamped Peaks Payload

2 Introduction

This document describes the formats of sensor data produced by the FAZT I4 1.0. There are two formats, one for wavelength peaks and the other for spectra of intensity versus wavelength. Each is transmitted in a TCP stream. Peak and spectral data are provided on different sockets. The number of streams provided on each socket is limited to one.

3 Overview

An I4 in operation periodically sweeps a laser across a range of wavelengths and measures the intensity of laser light received on each of four channels against wavelength. Measurements of the received intensity are provided in two forms, spectral and peak.

Spectral data provides all the information in a sweep in the form of intensity versus wavelength. Spectral data is suitable for diagnostics and specialist sensors, but because of bandwidth limitations, it can only be provided for a small fraction of sweeps – 16Hz for a single channel and 4Hz for all four channels.

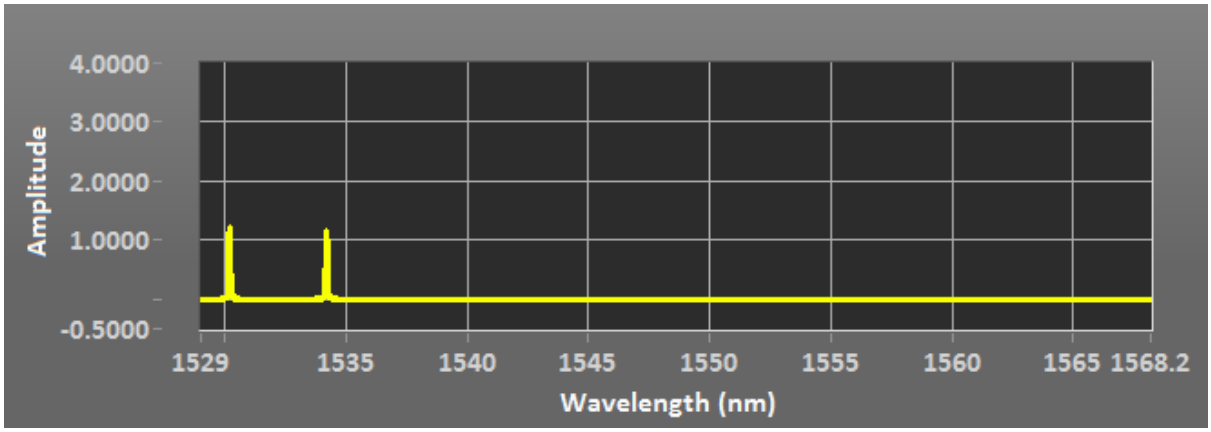

Figure 3-1: Spectral Data as displayed in FemtoSense

Peak data summaries the sweep as a number of peak wavelengths. Peak data is produced at the full sweep rate and is compact enough to represent a large number of peaks per sweep.

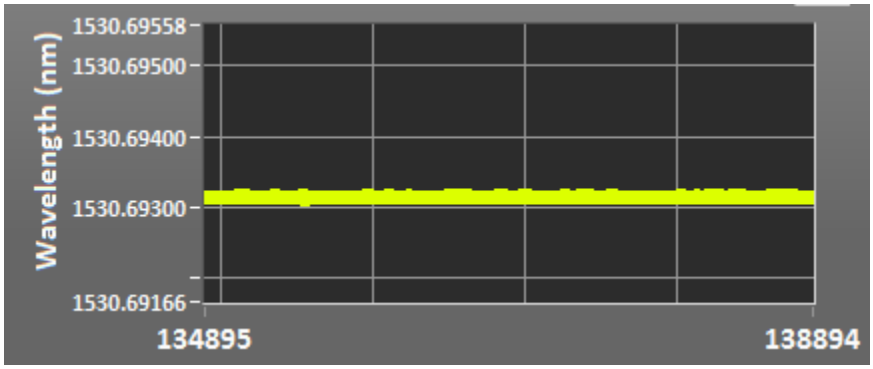

Figure 3-2: Peak Data as displayed in FemtoSense

The I4 is configured to sweep and detect peaks using the management API described in the FAZT I4 Management API Reference Guide.

4 Data Formats

The I4 produces a sequence of peak data 'packets' on TCP port 9931 and a sequence of spectral data 'packets' on TCP port 9932. Only one client is supported on each port. A second client will fail to connect.

The I4 transmits binary data on these ports and does not receive any data.

Packets are produced by the I4 at an exact rate determined by its configuration. Typical rates are 1 kHz for peak data and 4Hz for spectral data. If the client does not receive data as fast as it is produced, it is buffered for several seconds. After that, the oldest packets are discarded and a gap occurs in the data stream. The client can detect a gap by monitoring packet counters in successive packet headers.

Packets are encoded in binary with little endian byte order.

4.1 Generic Packet Format

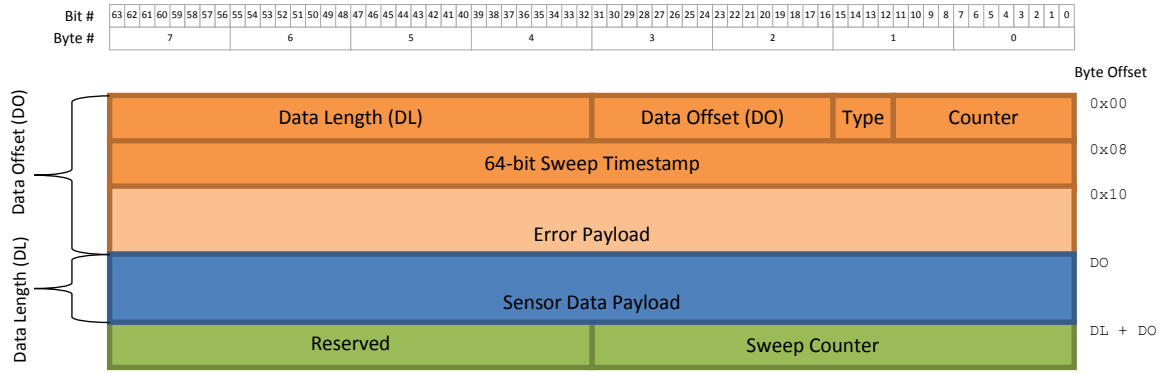

Figure 4-1: Sweep packet bit layout

All I4 data stream packets follow the general structure shown in Figure 4-1 above. The contents and format of the Sensor Data Payload differs depending on the Sweep Type (see 4.1.1, 4.2, 4.3 and 4.4).

The total Sweep packet size can be calculated as follows:

Total Packet size (bytes) = DO + DL + 8

Where 8 is the combined size of the 'Reserved' field and the 'Sweep Counter' field.

4.1.1 Offset 0x0: Counter / Sweep Type

- **Bits 11-0:** A 12-bit rolling "Packet Counter" which increments with every packet of the same type that is produced. The counter allows the client to detect missing sweeps in data stream
- **Bits 14-12:** A 3-bit "Sweep Type" flag, distinguishing between "Peak" (0x0), "Spectral" (0x1) and "Peak with timestamps" (0x2) sweep formats.

- **Bit 15:** A 1-bit "Triggered Mode" flag, indicating if the sweep was produced with an external trigger (0x1) or the systems internal trigger (0x0)

4.1.2 Offset 0x2: Data Offset

A 16-bit unsigned integer indicating the byte offset, from the start of the sweep packet, of the start of the peak data – e.g. 0x0010 indicates that peak data is located immediately after the 64-bit timestamp field (when no error information is included in sweep).

4.1.3 Offset 0x4: Data Length

A 32-bit unsigned integer indicating the length (in bytes) of the peak data at the offset indicated by the "Data Offset" field.

4.1.4 Offset 0x8: Timestamp

A 64-bit timestamp representing the time (in nanoseconds) since 1st January 1900 when the current sweep was produced by the I4.

4.1.5 Offset 0x10: Optional Errors

Optionally (see section 0) error information can be transmitted via the data stream. If present this is placed at offset 0x10. Each error is in a 64-bit field to perserve the 64-bit alignment of the peak data to follow.

4.1.6 Offset DO: Sensor Data Payload

The contents of the sensor data payload depends on the sweep type (see 4.1.1, 4.2, 4.3 and 4.4).

4.1.7 Offset DO + DL: 32-bit sweep counter

A 32 bit rolling sweep counter. This counter is incremented each time the laser sweeps.

4.1.8 Offset DO + DL + 0x04: Reserved

The final 32 bit field of the Sweep Packet is reserved for future enhancements. This is set to 0x0 in all packets.

4.2 Peak Payload

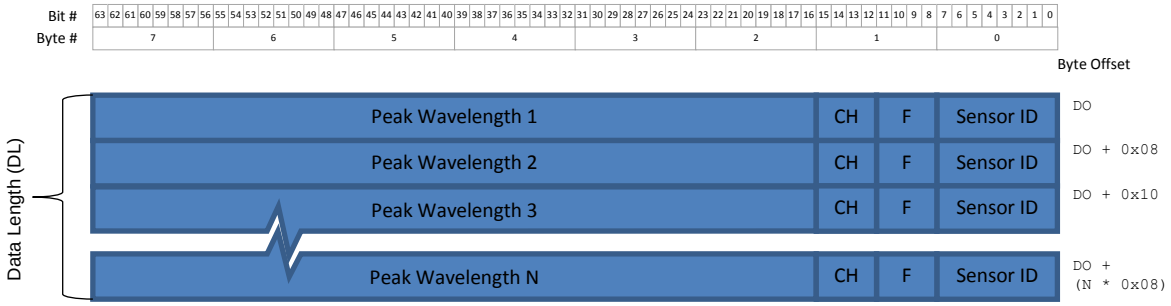

Figure 4-2: Peak Payload bit layout

Peak data is located immediately after the optional error data of the overall sweep packet. Each individual peak data is a 64 bits in total. The most significant 48 bits are the most significant bits of an IEEE 754 double precision number expressing the peak wavelength in meters. Truncating the number to 48 bits yields a precision of 0.014fm.

The remaining 16 bits identify the sensor using four bits for channel number, four bits for fibre number and eight bits for sensor number.

- **Bits 15-12:** Channel ID – 0 to 15
- **Bits 11-8:** Fibre ID – 0 to 15
- **Bits 7-0:** Sensor ID – 0 to 255

An example of parsing the Peak Payload can be found in the Appendix section 6.1.

4.3 Timestamped Peaks Payload

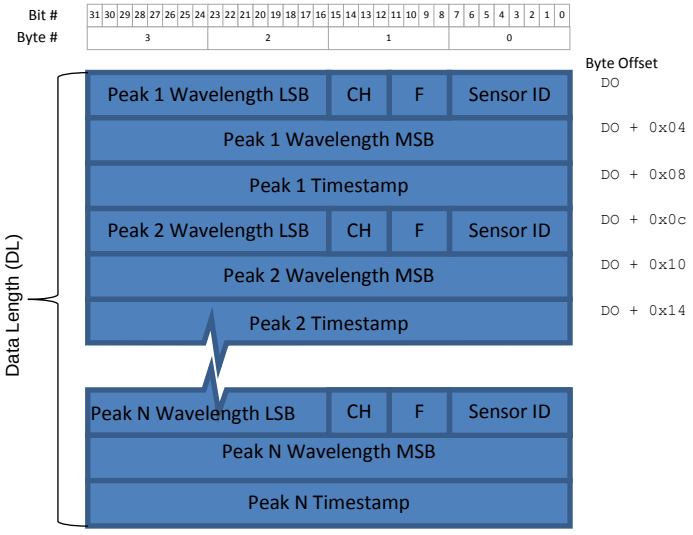

Figure 4-3: Timestamped Peaks Data Payload bit layout

The payload data is located immediately after the optional error data of the overall sweep packet. Each individual peak data is a 96 bits in total. The first (least significant) 64 bits follow the exact same bit layout as the Peak Payload described in 4.2. The following 32bits correspond to the sensor timestamp.

The sensor timestamp gives the relative time of the sensors value with respect to the sweep timestamp. The timestamp is a 32 bit unsigned integer in units of 0.5ns.

The 96bit pattern of topology/wavelength/timestamp is repeated for every sensor setup and in total constitutes the whole timestamped peaks payload data.

An example of parsing the Timestamped Peak Payload can be found in the Appendix section 6.2.

4.4 Spectral Payload

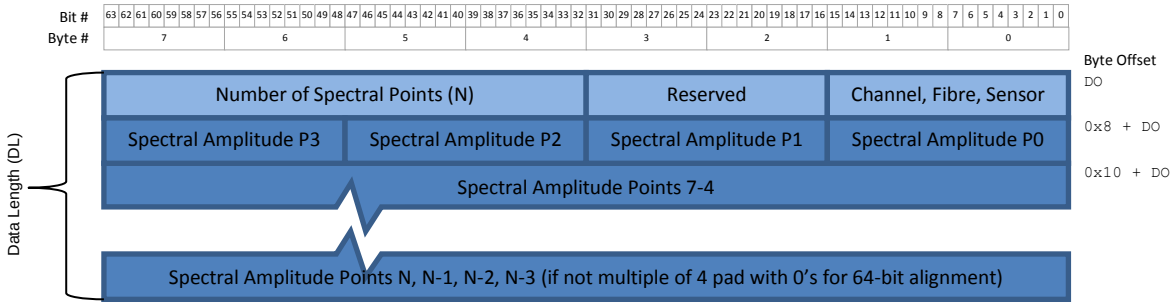

Figure 4-4: Spectral Data Payload bit layout

Spectral sweeps are formatted as shown in Figure 4-4 above.

FBG Spectral Sweep packets contain the following information (see 4.4.1 - 4.4.5)

4.4.1 Offset DO: A 16-bit Spectral data topology: Channel ID, Fibre ID, Sensor ID

- **Bits 15-12:** Channel ID – 0 to 15
- **Bits 11-8:** Fibre ID – 0 to 15
- **Bits 7-0:** Sensor ID – 0 to 255

4.4.2 Offset DO + 0x02: 16-bit Reserved field.

Reserved bits for future expansions. Should be ignored by client.

4.4.3 Offset DO + 0x04: 32-bit Number of spectral data values

Number of spectral data samples in the packet.

4.4.4 Offset DO + 0x08: Spectral Data

The N spectral samples as signed 16 bit integers:

Offset DO + 0x08: 16-bit fix point spectral sample P0,
Offset DO + 0x0A: 16-bit fix point spectral sample P1,
Offset DO + 0x0C: 16-bit fix point spectral sample P2,
Offset DO + 0x0E: 16-bit fix point spectral sample P3,
Offset DO + 0x10: 16-bit fix point spectral sample P4,
…….
Offset DO + DL – 0x0E: 16-bit fix point spectral sample PN-2,
Offset DO + DL – 0x0C: 16-bit fix point spectral sample PN-1,
Offset DO + DL – 0x0A: 16-bit fix point spectral sample PN

Values represent intensity on a relative scale. Small negative values may occur when intensity is low.

4.4.5 Offset DO + 0x08 + N*2: Spectral Data Alignment Fixup

Between 0 and 3 16 bit zero values to maintain 64 bit alignment

5 Error Reporting

I4 reports errors to the client application using two methods:

- Errors added to an error log accessible through the REST interface. See Faz I4 Management API Guide.
- Data related errors are reported by the I4 in the "Error Payload" fields of the Peak and Spectral packets.

5.1 Error Payload

All errors in the data stream are 64-bits in length with the following format:

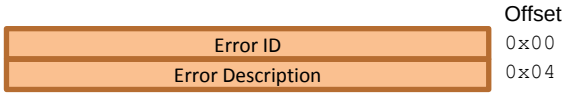

Figure 5-1: Error format

- **Error ID:** 32-bit identifier indicating type of error – e.g. Missing Peak, Critical System Error, etc.
- **Error Description:** 32-bit field providing additional error information – e.g. Channel, Fibre & Sensor location of Missing Peak

5.1.1 Data Related Errors

Data related errors are associated directly with the sweep data (peak or spectral) contained in a sweep packet (see section 4). I4 implementation of data related errors will include errors of the following types:

Error Type	Error ID	Error Description	Error Cause
Missing Peak	500	Channel, Fibre, Sensor info	***Error source:*** No peak was detected where one was expected from the interrogator configuration. Possible causes include: • Misconfiguration of sensor wavelength range or threshold • disconnected sensor
Multiple Peaks	501	Channel, Fibre, Sensor info	***Error source:*** More than the expected number of peaks was detected in the sensors wavelength range. Possible causes include: • Misconfiguration of sensor wavelength range or threshold

Internal Error	502-699	N/A	***Error source:*** Internal error Possible causes include: • Transient mismatch of configuration and data stream • Internal failure

Figure 5-2: Data Related Errors

Channel / Fibre / Sensor information is stored in the same 16-bit field described in section 4.4.1:

- **Bits 15-12:** Channel – 0 to 15
- **Bits 11-8:** Fibre – 0 to 15
- **Bits 7-0:** Sensor – 0 to 255

For “Missing Peak” and “Multiple Peaks” errors, the error field replaces the data field for the Peak in question. No wavelength is reported for the missing/multiple peak in the data field of the Peak Sweep packet (see section 4.2).

6 Appendix

6.1 C99 example code: parsing Peak Payload

```c
#include <stdio.h>
#include <stdlib.h>

typedef uint32_t peak_data_t[2];

/* extract wavelength value in meters */
double wavelength(const uint32_t* const peak_data) {
    uint32_t peak_value[2];
    peak_value[0] = (peak_data[0] & ~0xffff) | 0x7fff; /* mask out id bits */
    peak_value[1] = peak_data[1];
    return *(double*) &peak_value;
}

uint8_t channel_id(const uint32_t* const peak_data) {
    return (uint8_t) (peak_data[0] >> 12) & 0x0f;
}

uint8_t fiber_id(const uint32_t* const peak_data) {
    return (uint8_t) (peak_data[0] >> 8) & 0x0f;
}

uint8_t sensor_id(const uint32_t* const peak_data) {
    return (uint8_t) (peak_data[0] & 0xff);
}

int main(void) {

    peak_data_t peak_data;
    peak_data[0] = 0x47633201; /* channel 3, fiber 2, sensor 1 */
    peak_data[1] = 0x3EB9A701; /* ~1529e-9  = 0x3EB9A7014763FDE8 */

    printf("Sensor ID:\t%u\n", sensor_id(peak_data));
    printf("Fiber ID:\t%u\n", fiber_id(peak_data));
    printf("Channel ID:\t%u\n", channel_id(peak_data));
    printf("Wavelength:\t%.10e meters\n", wavelength(peak_data));
    return EXIT_SUCCESS;
}
```

6.2 C99 example code: parsing Timestamped Peak Payload

```c
#include <stdio.h>
#include <stdlib.h>

typedef uint32_t ts_peak_data_t[3];

/* extract wavelength value in meters */
double wavelength(const uint32_t* const peak_data) {
    uint32_t peak_value[2];
    peak_value[0] = (peak_data[0] & ~0xffff) | 0x7fff; /* mask out id bits */
    peak_value[1] = peak_data[1];
    return *(double*) &peak_value;
}

uint8_t channel_id(const uint32_t* const peak_data) {
    return (uint8_t) (peak_data[0] >> 12) & 0x0f;
}

uint8_t fiber_id(const uint32_t* const peak_data) {
    return (uint8_t) (peak_data[0] >> 8) & 0x0f;
}

uint8_t sensor_id(const uint32_t* const peak_data) {
    return (uint8_t) (peak_data[0] & 0xff);
}

/* extract time stamp value in seconds */
double time(const uint32_t* const peak_data) {
    return (double) peak_data[2] * 5e-10;
}

int main(void) {

    ts_peak_data_t peak_data;
    peak_data[0] = 0x47633201; /* channel 3, fiber 2, sensor 1 */
    peak_data[1] = 0x3EB9A701; /* ~1529e-9  = 0x3EB9A7014763FDE8 */
    peak_data[2] = 2000000; /* 1ms */

    printf("Sensor ID:\t%u\n", sensor_id(peak_data));
    printf("Fiber ID:\t%u\n", fiber_id(peak_data));
    printf("Channel ID:\t%u\n", channel_id(peak_data));
    printf("Wavelength:\t%.10e meters\n", wavelength(peak_data));
    printf("Timestamp:\t%f seconds\n", time(peak_data));
    return EXIT_SUCCESS;
}
```