CHILL/NCAR

Integrated Weather Radar Facility
IWRF

Proposed time series data format

DRAFT 6

Prepared by:

Mike Dixon, Chris Burghart, Joe Van Andel

NCAR

2009-10-20

1 Goal

The goal of this paper is to specify the radar time series format to be shared by the Integrated Weather Radar Facility (IWRF) of CSU-CHILL and NCAR.

2 Basic design – a serial stream of packets

The basic design of the time series format will follow that currently used by CSU-CHILL, though with significant modifications.

The data stream is made up of a serial stream of data packets, one following the other. Some packets will be sent more frequently than others. A pulse packet will be sent once for every pulse, i.e. at the pulse repetition frequency. By contrast, a radar_info packet will be sent relatively infrequently, say once per sweep or volume. A xmit_power packet could be sent say once per second, or however often the transmit power is sampled.

Most of the packet types contain only meta-data (commonly called header information). Only the pulse packets will contain data – they will have a meta-data header followed by the actual IQ data.

Each packet will contain an ID, as the first word, which will identify the packet type.

For example, the following packets may appear in the stream:

- radar_info
- scan_segment
- ts_processing
- xmit_power
- calibration
- pulse, pulse, pulse … pulse
- synchronization
- radar_info
- scan_segment
- pulse, pulse, pulse … pulse
- xmit_power
- pulse, pulse, pulse … pulse

The format will be the identical for TCP/IP transmission or storage on disk or other media.

3 Portability of the data types

All data types used in the format will be portable from platform to platform. Therefore,. the types must be independent of any specific computer language, such as C. For example, the C native types `int` or `float` are not acceptable because they are not inherently portable.

In this document, the following portable types are used:

type	byte width	description
ui08	1	8-bit unsigned integer
si08	1	8-bit signed integer
ui16	2	16-bit unsigned integer
si16	2	16-bit signed integer
ui32	4	32-bit unsigned integer
si32	4	32-bit signed integer
ui64	8	64-bit unsigned integer
si64	8	64-bit signed integer
fl32	4	32-bit IEEE floating point
fl64	8	64-bit IEEE floating point

Table 1: portable data type definitions

Use of unsigned integers will be limited to legacy formats only (such as RVP8), since Java does not support unsigned types.

4 Byte-order and swapping will be handled, as required, by the reader

We will make it the responsibility of the reader process to handle byte-swapping as required. The meta-data will contain sufficient information for the decision to be made on whether to swap or not.

The reason for this strategy is as follows: most hosts will probably be using a common byte order (little-endian) since they are Intel-based. Therefore, swapping into network-byte-order (big-endian) is an unnecessary use of CPU cycles.

However, if the data is read on a host which has a different byte order from the writer, byte-swapping will be required, and must be handled by the reader.

The byte order should be fixed in a data stream, i.e. it should not jump back and forth between big-endian and little-endian data.

5 Data rate, bandwidth, efficiency and meta-data size

Most data formats are a compromise between clarity and efficiency. Because the size of the IQ data dominates the overall size of the time-series data stream, we can afford to keep the design of the meta-data clear and easy to understand. In other words, we do not need to be 'clever' in the headers in order to save space.

To illustrate this, consider a typical dual-channel (dual receiver) radar storing data for 1000 gates. If we use 2-byte integers to store the IQ data, this will require 8 bytes per gate, i.e. 2 channels * 2 bytes * 2 fields. This means that the size of the IQ data for each pulse will be 8000 bytes.

If we precede the IQ data with a 256-byte meta-data header, for a total packet size of 8256 bytes, this will increase the total size by only 3%. This is an acceptable compromise for the sake of keeping the meta-data clear to understand and easy to maintain.

The size of the rest of the meta-data is trivial by comparison. Assuming that the radar pulses at a PRF of 1000, and that we send ALL of the other meta-data packets once per second, the pulse data rate will be 8200 * 1000 bytes/s, i.e. 8.2 Mbytes/s. All other meta-data packets together would require about 12 Kbytes/s. This will add only 0.15% to the data size.

6 Meta-data fields common to all packets

All packets will start with the following 56 bytes:

```
si32 packet_id;
si32 len_bytes;
si64 seq_num;
```

```
si32 version_number;
si32 radar_id;
si64 time_secs_utc; // secs since 1 Jan 1970
si32 time_nano_secs;
si32 reserved[5];
```

In the C/C++ implementation, this info is contained in the iwrf_packet_info_t structure.

The packet_id will be a number in the hex range 0x77770000 to 0x77770fff. The reason for choosing this range is as follows:

- it is a relatively unusual number range, so it will most likely be obvious if the reader is out-of-order with respect to reading the packets;

- the values can be effectively used as a test for byte order. If the ID does not fall within the expected range, try swapping. If the swapped number falls within the range, then swapping is required. If not, re-synchronization is required.

The len_bytes is the total length of the packet, i.e. it will include the common 56 bytes (iwrf_packet_info_t) at the start of the packet. For pulse packets only, this is the size of the meta-data plus the size of the following IQ data.

The version_number should start at 1, and will be used by software to make decisions, later on, as the format changes.

The radar_id is used to identify the relevant radar, if the data stream contains information from multiple radars, for example in dual-wavelength systems.

The time in seconds and nanoseconds is included so that all packets are time-stamped to a high accuracy.

The reserved values are included for later expansion as needed.

7 Packet size

The size of all of the packets (C structures) must be 2^n bytes where n >= 3. This allows for I/O efficiency and assures that all packets will be aligned at 8-byte boundaries.

8 Spare (unused) space in meta-data, and missing values

All meta-data structures should contain a generous number of unused spare fields, which can be filled out later as extra demands are added.

Missing integer values should be set to -9999.

Missing floating point values should be set to NAN.

9 Pulse packets

The pulse packets are the only packets which contain actual data. All other packets contain meta-data only.

A pulse packet is made up of:

- iwrf_pulse_header_t
- IQ data for channel 0
- IQ data for channel 1
- IQ data for channel 2
  ….....
- IQ data for channel (n_channels-1)

Within each channel, the IQ data is stored in ordered pairs, one pair for each gate:

I for gate 0, Q for gate 0
I for gate 1, Q for gate 1
….....
I for gate ngates-1, Q for gate ngates-1

The IQ data may be stored in a number of forms, as follows:

```
IWRF_IQ_ENCODING_FL32:
    4-byte floats, volts
IWRF_IQ_ENCODING_SCALED_SI16:
    voltages scaled as signed 16-bit ints
        volts = (si16 * scale) + offset
IWRF_IQ_ENCODING_DBM_PHASE_SI16:
```

```
    16-bit signed ints, storing power_dbm and phase.
        power_dbm = (si16 * scale) + offset
        phase = (si16 * 360.0) / 65536.0
IWRF_IQ_ENCODING_SIGMET_FL16:
    SIGMET 16-bit floats, see RVP8 documentation
```

If RVP8-specific meta-data is included, the normal pulse packet will be immediately preceded by the RVP8-specific header.

10 Synchronization packets

Every so often (say every 1 second) a synchronization packet will be sent. This will contain a series of bytes which are byte-order independent, and which can be used to re-sync with the data stream if needed.

The sync packet is made up of the 56 byte header, plus 8 synchronization bytes:

```
iwrf_packet_info
si32 0x2a2a2a2a
si32 0x7e7e7e7e
```

11 Calibration information

The current CHILL time series format contains calibration information spread over a number of different packet types.

We have simplified this to put all of the calibration information in a single packet, along with the radar information which is necessary for computing the radar constant. If this is not the best way to do it, we can go back to splitting the calibration information between packets.

12 Dual-polarization channel nomenclature

We have used the following terminology to distinguish between the various dual polarization channels:

- Hc = Horizontal co-polar, H receive, H transmit
- Hx = Horizontal cross-polar, H receive, V transmit
- Vc = Vertical co-polar, V receive, V transmit

- Vx = Vertical cross-polar, V receive, H transmit

This is a ‘receiver-centric’ way of naming the channels: the primary letter comes from the receiver channel, and the secondary letter determines whether it is co-polar or cross-polar.

13 IWRF namespace

In order to keep names unique, we add IWRF or iwrf to the start of most variable names.

14 Packet types

The following table lists the packet types and sizes.

Packet type	Size in bytes	ID in HEX
iwrf_sync	64	0x77770001
iwrf_radar_info	256	0x77770002
iwrf_scan_segment	4096	0x77770003
iwrf_antenna_correction	128	0x77770004
iwrf_ts_processing	256	0x77770005
iwrf_xmit_power	128	0x77770006
iwrf_xmit_sample	8192	0x77770007
iwrf_calibration	512	0x77770008
iwrf_event_notice	128	0x77770009
iwrf_phasecode	4096	0x7777000a
iwrf_xmit_info	128	0x7777000b
iwrf_ts_pulse	256 + IQ data size	0x7777000c
iwrf_rvp8_pulse_hdr	256	0x7777000d
iwrf_rvp8_ops_info	2048	0x7777000e

Packet types

15 Packet definitions

Complete packet definitions can be found in the header file *iwrf_data.h.*