BEAM

Blacknest BDS Data File Format

Preliminary – Live Document

Project	BDS
Date	2018-01-30
Reference	BdsDataFile
BDS File Format Version	1.2.0 (Initial format 1.1.0)
Author	Dr Terry Barnaby

Table of Contents

1. Introduction

Seismic sensor data is stored and transferred in a number of different formats. In order to make the BDS system as simple and flexible to use as possible the BDS system stores all data in its own internal format, BDS. This format has been designed so that it can encapsulate all of the information from external data formats. It is an internal format not intended for external use. This allows it to be easily modified and extended as required to support other external data formats or for system requirements.

Note that although all data file formats will store the seismic sensor data samples, most will not store all of the additional meta data information such as instrument responses, locations etc. The BDS data format can store the Meta data as well as the seismic sensor sample data. However, it is expected that only basic Meta data will be included in the BDS data files for consistency checking purposes. The BDS system stores the Meta Data in a database. A user or data program will need the seismic sensor data and the meta data information. The BDS API provides both the Meta data and the seismic sensor data.

In order to simplify data file access and allow the easy creation of data converters the BDS system has a data file access API. This API provides a simple, common access to seismic data files of any format including the BDS Data file format.

BEAM

This document describes the BDS Data File format that stores the seismic data. It is like SEED in some respects but the MetaData system is simpler and more flexible and the API is much simpler.

2. Features

- Keeps original data samples intact. No interpolation of original data sample values.
- Keeps original time stamps and block sizes intact.
- File and streaming support.
- In a file, fixed sized block based to allow for easy/quick search for data blocks over a particular time period.
- Variable sized packets to allow for varying number of samples per block, different data types and compression schemes.
- Free format ASCII MetaData attributes. This allows any set of MetaData to be stored in the file.
- Multiple streamlet support. Each streamlet can contain one or more channels.
- Data channels can be multiplexed at the channel level or sample level.
- Able to support synchronously or asynchronously sampled multiplexed data channels.
- Access to data while files are being created. Useful for real-time data access.
- "Canadian compression" support.
- 16bit integer, 32bit integer and 32bit floating point sample format support. (Could be easily extended).
- Simple API to read/write file files.
- Checksum on the packet level.
- Sequence numbers so that missing packets can be identified.
- Easily extendible for future requirements.
- Time stamps at microsecond accuracy.
- Can store blocked and timestamped ASCII log data
- Ability to append data to existing files for use when streaming data into day files with backfill.

3. Overview

A data file or stream consists of a number of variable length packets of data. There can be a number of different packet types. At the moment just Info, InfoExtra and Data packets are defined. In the case of a file these packets are stored in fixed sized blocks, by default 64KBytes, to allow easy searching for data for a particular period. In a stream the bare packets are sent.
There is the concept of a streamlet and a sequence number within the streamlet. This allows multiple streamlets of data to be encapsulated within the file. Normally each data channel is stored in a separate streamlet although sample-multiplexed sets of data channels can be stored within one streamlet.
The Data packet size can be made the same as or a multiple of the original data's block size to synchronise time stamps if required. Larger packet sizes will improve compression efficiency and speed up data searching and access. All binary data is in little-endian format and can be readily converted to big-endian format.

All Packets have the following binary header:

Item	***Type***	***Description***
type	UInt32	The type of this packet
length	UInt32	The length of the packet in bytes

streamlet	UInt32	The streamlet number of this packet
sequence	UInt32	The stream packet sequence number
checkSum	UInt32	Block checksum (header and data - checksum field)
startTime	TimeStamp	The start time. At microsecond accuracy
endTime	TimeStamp	The end time. At microsecond accuracy

Info Packets are interleaved with the Data packets and can be repeated at intervals so that the BdsData can be used in a streaming system. The channel data blocks can be grouped together so that the data for a set of channels over a particular time period could be easily retrieved. When Meta Data changes a new set of Info blocks would be sent for the new time period.

Two main forms of the BDS data file format have so far been defined: BDS-SM and BDS-CM. BDS-SM is a sample multiplexed format. In this case all the channels samples are multiplexed together into blocks of data. The samples have to be synchronously sampled in order for this to work. BDS-CM is a channel multiplexed format. In this case each channels samples are separately stored in individual blocks of data.

4. Time Stamp

The time stamp consists of the following fields:

Item	*Type*	*Description*
year	UInt16	The year
yearDay	UInt16	The day of the year
hour	UInt8	The hour
minute	UInt8	The minute
second	UInt8	The second
spare	UInt8	Spare padding entry
microSecond	UInt32	The microsecond field

A Time Stamp with the year set to 0 means undefined.

5. Info Packets

An Info or InfoExtra Packet would have the following format:

Item	*Type*	*Description*
type	UInt32	The type of this packet
length	UInt32	The length of the packet in bytes
streamlet	UInt32	The streamlet number of this packet
sequence	UInt32	The stream packet sequence number
checkSum	UInt32	Block checksum (header and data - checksum field)
startTime	TimeStamp	The start time. At microsecond accuracy
endTime	TimeStamp	The end time. At microsecond accuracy

numItems	UInt32	The number of items
items[]	Item	A list of items

Normally Info packets are stored in streamlet 0.

Each item has the following format:

Item	*Type*	*Description*
nameLen	UInt32	The length of this entry
nameStr[]	UInt8	The Item's name in null terminated ASCII
valueLen	UInt32	The length of the value field
valueStr[]	UInt8	The Item's value in null terninated ASCII

The "itemName" field has a hierarchal naming scheme using the "." character as a separator. Arrays of items uses the "[n]" format.
(Note: We could have a type field here so that binary data, such as pole/zero frequency responses, could be stored in binary form.)
InfoExtra packets are designed for additional MetaData such as error lists that could be large and would not normally be needed when accessing data.

6. Data Packets

A Data packet has the following format:

Item	*Type*	*Description*
type	UInt32	The type of this packet
length	UInt32	The length of the packet in bytes
streamlet	UInt32	The streamlet number of this packet
sequence	UInt32	The stream packet sequence number
checkSum	UInt32	Block checksum (header and data - checksum field)
startTime	TimeStamp	The start time. At microsecond accuracy
endTime	TimeStamp	The end time. At microsecond accuracy
numChannels	UInt32	The number of channels
numSamples	UInt32	The number of samples
channelNum	UInt32	The number of the first channel within this data block
segmentNumber	UInt32	The segment number of this data block
packFormat	UInt8	The Sample packaging format: Sample Multiplexed, Channel Multiplexed Canadian Compression
sampleFormat	UInt8	The sample format: Int16, Int32, Float32
info	Name/Value pairs	Packet Meta Data
data[]	UInt8	The raw data

Either a single channels data or data for multiple, synchronously sampled, channels can be stored in this packet.
The raw samples can be of several different sample formats including: Int16, Int32 and Float32. The data can be compressed using a number of methods although we will only support Canadian initially. The number of samples per data packet could be set at the same or a multiple of the original data's block size to eliminate issues with time stamp interpolation.
The Info field is use to store data block based meta data. This is used to store ASCII LOG data in the "log" field. It is also used for the TapeDigitiser data which has information such as the FM Signal levels for each channel. The format of the info field is a list of name/value pairs in ASCII. The binary representation is as follows:

Item	*Type*	*Description*
number	UInt32	The number of following name/value pairs
nameLen	UInt32	Length of the following name string
nameStr[]	UInt8	The ASCII name field, null terminated
valueLen	UInt32	Length of the following value string
valueStr[]	UInt8	The ASCII name field, null terminated
		... The next Name/Value pair ...

The structure of the data in the packet depends on the format.
For non compressed data it consists of a simple two dimensional array of data. The first dimension is the channel number the second is the sample number. Thus each channels set of data is contiguous.

7. Standard Info Packet Entries

Although the Info Data is free format ASCII, there are some standard definitions. The following shows some examples of these.
The full set of definitions are defined in the BdsMetaData document.

Item Name	*Description*
bds.version	The BDS File Version
bds.format	The BDS format (BDS-SM or BDS-CM)
startTime	The start time of the following data.
endTime	The end time if a file format (not in streamed data)
array	The Array the data is from if only from a single array
description	Some description
channels.number	The number of channels
channels.synchronous	A boolean defining if the channels are synchronously sampled
channel1.startTime	The start time
channel1.endTime	The end time

channel1.network	The network the data is from
channel1.station	The station the data is from
channel1.channel	The channels identifier name
channel1.source	The data source, "Master" is the normal.
channel1.type	The channels type (BHZ etc)
channel1.auxId	The auxillary or loaction ID.
channel1.sampleRate	The channels sample rate
channel1.sampleFormat	The sample format Int16, Int32, Float32 etc
channel1.streamlet	The streamlet this channels data is in. (Multiple channels will share a streamlet in Sample Multiplexed mode)

8. File Blocks

The BDS Data Packets would be stored within fixed sized blocks in a file for quick and easy random access to the data. A BDS Data block has the following format:

Item	***Type***	***Description***
type	UInt32	The type of this block (Magic number)
length	UInt32	The length of the complete block in bytes
packetOffset	UInt32	The offset in bytes to the next packet header within the block
data[]		The raw packet data

9. Appending Data

The file data is quantised into blocks of a fixed length (default 64 kBytes). Each block contains multiple packets of information and data. The last packet written in a file write or append session will have a packet type of 0. The first packet written when a file is created or appended to will be an information packet. This will contain the channel to streamlet mapping amongst other information.

A BDS format data file can be opened in append module (“a+”). You can only append data to channels the file was originally created for as setup by the setInfo() API call. If you try and append other channels, the setInfo() call for this will fail.

When data is appended to a BDS file it is added into a new packet following the previously last packet in the file. The data added must be for the same data channels (Network:Station:Channel:Source) as defined in the files first info packet. The first new packet added will be an info packet matching the files main info packet. The time order of data blocks is unimportant as sorting is automatically performed when the file is read.

10. Notes

1. The Info Packet facility could provide the ability to include all of the Seismic Meta Data for the period in question, if required, including instrument responses etc.
2. Canadian Compression not fully implemented or tested. The code is in here as an example.
3. When creating sample-multiplexed data we compress each channel independently and then store

the compressed blocks within the one packet.

4. Can only handle one CM channel or one set of SM channels. The streamlet allocation is hard-coded to the data channel number for writes.
5. Currently allows data to completely overlap in time when appending to the file.