BEAM

Blacknest BDS Development

Programming Manual – 2.1.4

Project	Blacknest
Date	2018-10-12
Reference	blacknest/bdsDevelopment
Author	Dr Terry Barnaby

Table of Contents

1. References

- The BEAM Blacknest support website at: https://portal.beam.ltd.uk/support/blacknest. This provides detailed information on the BDS system and the current AutoDRM, information on alternative AutoDRM implementations and information on data formats.

2. Introduction

This document provides information on developing programs for the BDS system. Yours should first read the document: BdsUserManual.pdf that provides an overview of the system and how it functions.

There is also more information in the BDS Software Development presentation BdsDevInfo.odp.

The BDS C++ low level API documentation is provided at bdsApi as well as some examples.

3. Overview

This section covers the overall concepts of the API. 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 has a generalised data access API that allows access to any data format.

The seismic and other data are stored in binary files on the BDS server platform and we refer to these as the Data or Seismic data. This data may have some low level Metadata included in the binary files.

Metadata information on the stations, channels, responses etc. is stored in a MySQL database. This

Directors: Dr T.K. Barnaby, M.J. Thomas
Registered Number: 2415065
Registered office: Beam Ltd, Northavon Business Centre, Dean Road, Yate, Bristol BS37 5NH, UK.
VAT Number: 520-2521-05
Web: www.beam.ltd.uk
Email: info@beam.ltd.uk

database also provides information on the data stored in the data files.

All of this data and information is accessed through the, gate keeper, BdsServer daemon program via a set of network API's.

The BDS system has an overall BdsServer daemon program running as the gate-keeper to all functionality. This is accessed via an object orientated binary API. Various BDS client programs use this API to manipulate the Data and Metadata of the BDS system. As well as providing API functions for Data and Meatadata access there are functions for login authorisation and overall system management.

The BdsApi has been developed using the BOAP (BEAM Object Access Protocol). This provides a simple but powerful Object Orientated RPC mechanism. The BdsApi is written in a high level interface definition language (IDL). The bidl tool generates the client and server side 'C++' interface and implementation files for the API. These are then provided as a set of 'C++' header files and a binary library file for the clients to link to. The BOAP system employs a simple BOAP name server process that provides a translation between object names and IPAddress/Socket numbers. The BOAP name server runs on the BDS Controller. More information on the BOAP system can be found in the libBeam documentation.

THE BDS API implements a number of data storage classes and three interface objects. The interface objects are:

1. **Bds::DataAccess** BDS Data API: This will provide read only access to the data and meta data. It will be used by the AutoDRM email and Web systems as well as for program access to the data.
2. **Bds::DataAddAccess** BDS DataAdd API: This will provide read and restricted write access to enable the adding of data to the system. It will not allow deletions of data to be performed. It is designed to be used by manual and automatic data adding programs.
3. **Bds::AdminAccess** BDS Admin API: This will provide full read/write access to the data and meta data as well as administrative configuration information.

These API's share the same functions with the Bds::DataAccess API have the minimal set of read-only functions and the BDS:AdminAccess API having all of the possible functions.

All BDS functions are accessible via these API's, given a suitable user/password is available, to any client program. The BDS API is provided as a C++ object orientated library that makes use of the standard BeamLib library of classes and functions.

There is also a Python version of this library available, that sits on top of the C++ API library to allow Python programs to directly access the BDS system. Details of this are listed later in this document.

3.1. Seismic Data

The BDS DataAPI is based on the following common data file model.

- Seismic sensor data is split into individual channels each containing a single data stream.
- Data files or streams may contain multiple channels of data.
- Each channel has a set of meta data associated with it. This defines things like the StartTime, EndTime, Network, Station, Channel, SampleRate, CalibrationFactor etc.
- The data for each channel is split into segments. Each segment may be for a different period of time and/or a different network or source.
- Each channels segment data is split into blocks. These may be variable or fixed length. Each block has start and end time stamps associated with it. The BDS keeps the original sources blocks intact

Directors: Dr T.K. Barnaby, M.J. Thomas
Registered Number: 2415065
Registered office: Beam Ltd, Northavon Business Centre, Dean Road, Yate, Bristol BS37 5NH, UK.
VAT Number: 520-2521-05
Web: www.beam.ltd.uk
Email: info@beam.ltd.uk

as to start, end times and physical data etc.
- Each data block may have special meta data information associated with it depending on the original data format.
- If there are multiple channels, these may be synchronously sampled or independently sampled.
- Data can be multiplexed by sample or by channel.
- Raw data has an original sample format of Int16, Int32 or Float32. This can be extended in the future.
- The BDS system allows a set of channel data, from the same time period, to be returned as a single set of sample multiplexed data if the channels are synchronously sampled.

3.2. Selecting Data or Metadata

In order to access Data or Metadata from the BDS system, the set of channels required needs to be selected. The BDS system allows for the selection of multiple channels of data from various sources over a given time period. To achieve this the BDS system is passed a Selection object. The Selection object has the following main fields:

Item	*Description*
startTime	The Start time to the nearest micro-second
endTime	The End time to the nearest micro-second
channels	The List of SelectionChannel Objects

Each SelectionChannel has the following fields:

Item	*Description*
network	The network the data is from
station	The Array or Station name
channel	The Channel name
source	The data source (Master, Tape, Processed etc)

Any number of SelectionChannel entries can be included. Each attribute can be set to a null string, which is used as a synonym for any, or a suitable value for selection. Standard regular expression characters, such as the wild card character "*" and “^[a-z]*?.*” etc. can be used in any of the fields. The BDS system will expand the given set of SelectionChannel objects to a set defining unique channel data or metadata segments in the system.
The data selection scheme will return a DataInfo object describing the data selected. This will contain a number of separate channels of data. Each channel can have multiple segments of data. Segments of data are based on time periods where the data is split into multiple files or where there are multiple sets of data from different data sources (for example Digital and/or Tape).
The Metadata selection scheme will return a set of suitable Metadata matching the selection criteria. This may be stations, channels, responses etc.

4. The API

The BDS API is heavily object orientated and based on the BeamLib set of C++ classes. It is documented in: libBeamApi.

Directors: Dr T.K. Barnaby, M.J. Thomas
Registered Number: 2415065
Registered office: Beam Ltd, Northavon Business Centre, Dean Road, Yate, Bristol BS37 5NH, UK.
VAT Number: 520-2521-05
Web: www.beam.ltd.uk
Email: info@beam.ltd.uk

The core C++ classes used are:

Class	*Description*
BString	A variable length string class used for storing, passing and manipulating ASCII text strings.
BError	Most functions return a BError object to provide the status of the functions operation. A BError object has a number and a string. The error numbers are listed in the API with 0 indicating Ok. The string is a human readable error message.
BList<Type>	This is a generic doubly linked list object that can be typed to store any other C++ object.
BArray<Type>	This is a generic contiguous memory array object that can be typed to store any other C++ object.
BTimeStamp	A date/time to microsecond resolution

5. BDS Data Access

The BDS Data Access API provides a simple set of functions that allow access to the data within the BDS system. The API allows the user to select a set of data channels and then stream them over either using the BDS data block API or in one of the formats that the BDS system has data converters for.

The BDS Data Access API has the following core functions:

Function	*Description*
BError getSelectionInfo(SelectionGroup group, SelectionInfo selectionInfo)	Returns information on all the Networks, Stations, Channels and Sources that the BDS system knows about. Useful for GUI driven data selectors.
BError getSelections(SelectionGroup group,Selection selectionIn, Selection& selectionOut);	Expands the given selection to match the data contents of the BDS system
BError dataSearch(Selection selection,DataInfo& dataInfo);	Searches for data matching the given selection and returns information on the associated data channels.
Berror dataGetChannelInfo(DataInfo dataInfo, ChannelInfos* channelInfos);	Returns the channel MetaData in structured form

Directors: Dr T.K. Barnaby, M.J. Thomas
Registered Number: 2415065
Registered office: Beam Ltd, Northavon Business Centre, Dean Road, Yate, Bristol BS37 5NH, UK.
VAT Number: 520-2521-05
Web: www.beam.ltd.uk
Email: info@beam.ltd.uk

BError dataOpen(DataInfo dataInfo, String mode, String format, UInt32 flags, DataHandle& dataHandle);	Opens a data stream. This will open a data stream which will contain all of the channels listed in dataInfo. The mode will be "w" for writing data and "r" for reading data. The format will be one of: API, BDS, BKNAS, IMS1.0 etc. API defines the BdsApi access scheme. The flags argument gives options such as return fullblocks etc
void dataClose(DataHandle dataHandle)	Closes the stream
API Stream Write Interface	
BError dataSetInfo(DataHandle dataHandle, DataInfo& dataInfo)	Provides information on the data. This is used to create file headers etc.
BError dataPutBlock(DataHandle dataHandle,DataBlock& data)	Writes a data block to the file. These have to be sequential.
API Stream Read Interface	
BError dataGetInfo(DataHandle dataHandle, UInt32 infoExtra, DataInfo& dataInfo)	Gets information on the data from the files data header and perhaps the data blocks. This will include the real sample rate and actual number of samples.
BError dataGetWarnings(DataHandle dataHandle, BStringList& warnings);	Return a list or warnings.
BError dataSeekBlock(DataHandle dataHandle, BUInt32 channel, UInt32 segment, BTimeStamp time, BUInt32& blockNumber)	Seeks to a particular data block given a time. This can operate on a single channel if a channel number is given or on multiple channels if channel number is 0. The segment parameter is the data segment number.
BError dataGetBlock(DataHandle dataHandle, BUInt32 channel, UInt32 segment, BUInt32 blockNumber, DataBlock& data);	Reads a data block from the file. If the channel number is given, then it reads data from a given channel. If the channel number is 0 it will read a set of data from all of the channels in sample multiplexed form if this is possible. The segment parameter is the data segment number.
Formatted Stream Read Interface	
BError dataFormatedRead(DataHandle dataHandle,UInt32 number, Array<UInt8>& data)	Returns the next set of bytes of pre-formatted data. For example BKNAS, IMS etc.
BError dataFormattedGetLength(DataHandle dataHandle,UInt64& length);	The total length in bytes of the formated data.

Directors: Dr T.K. Barnaby, M.J. Thomas
Registered Number: 2415065
Registered office: Beam Ltd, Northavon Business Centre, Dean Road, Yate, Bristol BS37 5NH, UK.
VAT Number: 520-2521-05
Web: www.beam.ltd.uk
Email: info@beam.ltd.uk

5.1. BDS DataBlock

When reading/writing data from/to a BDS stream there is a single Object called a BDS DataBlock that stores the data. This object has the following attributes:

Atrribute	*Description*
startTime	The Start time
endTime	The End Time
channelNumber	The channel number. 0 if all channels
segmentNumber	The segment number. 0 if all segments
channelData	Two dimensional array of data. Each channel can have a different number of samples
info	Extra meta data information. This may be available with particular data formats such as the TapeDigitiser

6. Metadata Access

The BDS stores seismic Metadata information in a MySQL database. The BDS API provides a generic API to access this data. There are a set of API functions per Metadata type. For example for Station information there is the following functions:

Function	*Description*
BError stationGetList(Selection sel, BList<Station>& stations)	Returns a list of all Station objects that match the selection criteria.
BError stationUpdate (BInt32 append, Station station, BUInt32 &id)	Either updates a Station entry in the database or adds a new entry. The id of the entry is returned.
BErrorstationDelete (BUInt32 id)	Deletes the Station entry

There are a set of functions for each Metadata type.

7. Python API

The BDS Python API is built on top of the standard BDS 'C++' API using the SWIG API generator. Thus all of the standard BDS C++ API documentation applies however there are some differences due to the language facility and syntax differences.

The Python language is interpreted rather than compiled and does not require strict types like 'C++'. This can help speed up the development of simple tools and programs, but it can result in less robust and less maintainable code.

To use the BDS API library import the “bdslib” or “bdslibe” modules. The “bdslibe” provides an exception based API as noted below.

The SWIG system wraps the BDS C++ objects in a Python object layer. Your can then ineract with the C++ BDS objects from Python in the same way as you would have done in C++ apart from a few

Directors: Dr T.K. Barnaby, M.J. Thomas
Registered Number: 2415065
Registered office: Beam Ltd, Northavon Business Centre, Dean Road, Yate, Bristol BS37 5NH, UK.
VAT Number: 520-2521-05
Web: www.beam.ltd.uk
Email: info@beam.ltd.uk

differences. The Python API is the same as the C++ API apart from a few coding style differences.

1. Reference returns: 'C++' allows references/pointers to be passed as function arguments which allows functions to return values. Python does not support this. Instead Python provides the ability for functions to return multiple items on the left-hand side. When using a BDS API call that in 'C++' returns items by reference, the Python equivalent will have these returned on the left hand side. For example:

   ```
   err = bds.channelGetList(selection, channels);        // C++

   (err, channels) = bds.channelGetList(selection);      # Python
   ```

2. Testing for error returns. All BDS API calls return a BError object. This provides information as to if the function completed successfully or if there was an error. The BError object contains both an error number and an error string. 'C++' allows an “if” statement to have an assignment operator. This makes returning and checking errors quite concise. Python does not allow this and requires a separate assignment and if statement. For example:

   ```
   if(err = bds.channelGetList(selection, channels)){
           return err;
   }

   (err, channels) =  bds.channelGetList(selection);
   if(err):
           return err;
   ```

3. Exceptions: The BDS API does not use 'C++' exceptions. As the BDS functions all return a BError object we have provided an alternative Python API library that uses exceptions for all BDS API calls instead of returning a BError object. With this calls can be written thus:

   ```
   try:
           channels = bds.channelgetList(selection);
   except ExceptionBError as e:
                   print "Execption", e.number, e.string;
   ```

There are also a few “gotchas”:

1. In C++ parameters can be passed too and from functions by reference/pointer or by value (a copy). In Python everything is passed by reference but a reference counter is provided on each object to make sure they are kept around as long as there is an active reference to them. The SWIG Python interface wraps the C++ objects in a Python type object and manages reference counters at this level as Python would normally do. However if you return a part of a C++ object from a function, perhaps an embedded BList, then the underlying C++ object could disappear when the top level C++ object goes out of scope. So if returning a portion of a C++ object you will need to make a deep copy of the object you are returning. There is no direct means of making a deep copy of the object however, you will have to manually do this. Another option is to set the “thisown” parameter on the toplevel object to 0. This keeps that object in memory. It is a memory leak however.

Directors: Dr T.K. Barnaby, M.J. Thomas
Registered Number: 2415065
Registered office: Beam Ltd, Northavon Business Centre, Dean Road, Yate, Bristol BS37 5NH, UK.
VAT Number: 520-2521-05
Web: www.beam.ltd.uk
Email: info@beam.ltd.uk

8. Examples

There are a number of BDS C++ and Python API examples in the /usr/bds/bdsExamples directory.

Directors: Dr T.K. Barnaby, M.J. Thomas
Registered Number: 2415065
Registered office: Beam Ltd, Northavon Business Centre, Dean Road, Yate, Bristol BS37 5NH, UK.
VAT Number: 520-2521-05
Web: www.beam.ltd.uk
Email: info@beam.ltd.uk