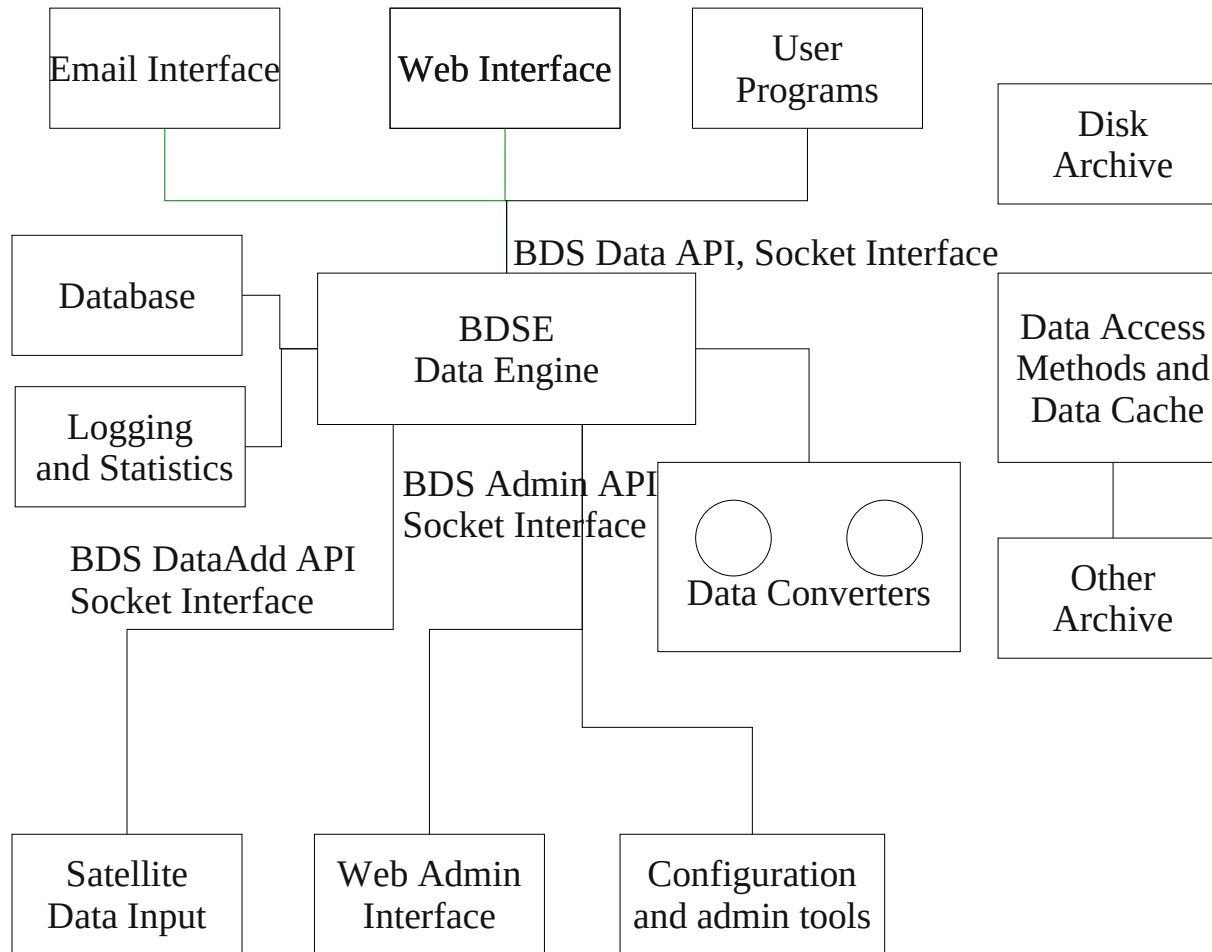# BDS Software Development

- BDS - Blacknest Data System

- System to store and recover seismic sensor data together with Meta data.

- Provide an introduction to programming in the BDS environment

- Requires 'C' and 'C++' development knowledge

BDS Overall Design

BDS Overall Design

BDS Software areas

We will cover:

- C++ Programming
- Client Software
- Scripts
- Data Converters
- Server Development
- API modifications

Programming Languages

- C++ is the primary development language
- Unix shell (BASH) used for basic scripts
- Python is used in some of our test programs
- PHP is used in the example web client
- BDS API can be ported to other languages

C++ Programming

- Based on the 'C' language

- Has Object Orientated extensions

- Other extensions including: Templates,References,Default Arguments, Variable hiding,Read Only, Name Spaces, Stricter compiler

- If used in a good Object Orientated style code can be written at a higher level of abstraction and more closely match the problem domain and hence be easier to understand.

- If used badly can easily create unmanaged code.

C++: Classes and Objects

- Description of a new type
- Can be low level types such as Complex, String, List or higher level such as BdsChannel
- "struct" like semantics with list of member functions. (Has data and function members)
- Class declaration (Interface) in header .h file
- Class definition (Implementation) in .cpp file

C++ Complex number class Definition

```
class Complex {
public:
                    Complex(double real = 0.0, double imag = 0.0);

        void        setReal(double real);
        void        setImag(double imag);

        double      getReal();
        double      getImag();
        double      abs();

        Complex     operator+(Complex c);
        Complex     operator*(Complex c);
private:
        double      oreal;
        double      oimag;
};

Complex     c1, c2(3.0, -4.1);

c1 = c1 * c2;
printf("Complex: %f, %f\n", c1..getReal(), c1.getImag());
```

C++ Complex Class Implementation

```
Complex::Complex(double real, double imag){
    oreal = real;
    oimag = imag;
}

double Complex::getReal(){
    return oreal;
}

void Complex::setReal(double real){
    oreal = real;
}

Complex Complex::operator+(Complex c){
    Complex      r;

    r.oreal = oreal + c.oreal;
    r.imag = oimag + c.oimag;
    return r;
}
```

References and const

```
Complex operator+(Complex c){
    r. oreal = oreal + c.oreal;
}

Complex operator+(Complex* c){
    r. oreal = oreal + c->oreal;
}

Complex operator+(Complex& c){
    r. oreal = oreal + c.oreal;
}

Complex operator+(const Complex& c){
    r. oreal = oreal + c.oreal;
}

Complex operator+(const Complex& c) const {
    r. oreal = oreal + c.oreal;
}
```

BEAM Class Library

- C++ is used in an Object Orientated Style

- The BEAM class library is used for basic types such as:

    - Strings, Lists, Arrays, BOAP RPC, Threads and Locking, Network I/O, Database I/O, Debug

- Avoidance of pointers at higher level interfaces

- Uses CamelCase naming convention. Upper case first letter for types and constants. Lower case first letter for functions and parameters.

Beam Class Library: Strings

- Variable length on heap

- Efficient shared reference counted string

BString   s1;

BString   s2;

s1 = s1 + s2;

if((s1 == s2) || (s1 > s2)){

}

s1 = s2.subString(0, 3);

s1 = stringToUppercase(s2);

cout << s1 << "\n";

printf("%s\n", s1.retStr());

Beam Class Library: Arrays

BArray<BString>   a;

a.setSize(64);

a[32] = "Hello";

for(int i = 0; i < a.size(); i++){

printf("Position: %d Value: %s\n", i, a[i].retStr());

}

a = functionAppend(a, a);

References:

BArray<BString> functionAppend(BArray<BString>& a1, Barray<BString>& a2);

Beam Class Library: Lists

```
BList<int>    l;
BIter         i;
l.append(1);
l.append(2);
for(l.start(i); !l.isEnd(i); l.next(i)){
printf("Value: %d\n", l[i]);
}
```

Beam Class Library: Error

- BError class is used to return errors from functions.

- BError has an error number and a string.

- Can be used in "if" statements and passed back up the calling function tree.

- Can be passed through exceptions.

BError   err;

if(err = func()){

cerr << "Error: " << err.getString() << "\n";

}

BOAP

- Beam Object Access Protocol
- Provides simple to use object RPC mechanism
- Uses an IDL language "compiled" with bidl.
- Produces a 'C++' Client and Server side API library.
- Can produce API libraries for other languages

BOAP: Example IDL

```
module Bds {
apiVersion = 4;
enum Errors      { ErrorOk, ErrorMisc, ErrorWarning };

class Point {
    Float64        x;
    Float64        y;
};

interface AdminAccess {
    Error      connect(in String user, in String password);
    Error      setUser(in String user, in String email);
    Error      setUserReal();
    Error      getVersion(out String version);
};
};
```

BOAP: Example Client Use

```
DataAccess      bds;
BError          err;
BString         version;

if(err = bds.connectService("//hostName/bdsDataAccess")){
    // Some error handling
}

if(err = bds.connect("test", "beam00")){
    // Some error handling
}

if(err = bds.getversion(version)){
    // Some error handling
}
cout << "Version: " << version << "\n";
```

BDS Development Libraries

- BdsLib: Main BDS API library. Primarily the BDS, BOAP based, API with some additional helper functions. "BDS" name space.

- BdsDataLib: BDS data access library. Includes seismic data converters. "BDS" name space.

- TapeDigitiser: Library from Tape digitiser project.

- BeamDsp: BEAM Digital Signal Processing library for TapeDigitiser import.

- Gcf: Standard GCF library

BDS Make environment

- The BDS build system uses a simple "Makefile" based system.

- There is a master "Makefile.config" file at the root of the software development tree that defines the standard build parameters.

- Each directory has its own Makefile to build its code that includes the Makefile.config. Tree of Makefiles

- The main make targets are:

  - all: The default to build the code

  - install: To build and install the code (DESTDIR is destination)

  - clean: To clean the code

  - rpm: To make an RPM package (At the higher levels)

BDS Client Programming

- Uses the BDS API library for access to the BDS Server.

- Simple to use (I hope !)

- Three access Objects: DataAccess, DataAdd, AdminAccess

- API Version

- Note requires userId/password for access so the connect() call has to be used first.

- Some examples in bdsExamples

Seismic data manipulation

- Fairly simple API to access seismic data and read meta-data.

- Data format independent. Two access methods: Direct data access and Formatted data access.

- dataOpen(), dataGetBlock(), dataFormattedRead(), dataClose()

- dataGetChannelInfo() gets all metadata associated with the data.

- Data selection scheme by Selection class that uses: StartTime and EndTime and a set of Network, Station, Channel and Source settings.

- Look at: bdsDataClient1, bdsDataClient2, bdsDataClient3

- Look at: Selection class

Metadata manipulation

- Can read, append and update meta-data.

- API allows sets of meta data items to be accessed via individual calls.

- Selection scheme by Selection class that uses: StartTime and EndTime and a set of Network, Station, Channel and Source settings. As per data access.

- All changes validated and tracked in BDS Changes database and synchronised with backup system. (Validation could be improved)

- Look at bdsMetaData1.cpp

Client Scripts

- Scripts in bash/csh/python/perl etc

- Can use BDS command line programs: bdsDataAccess, bdsImportData etc

- bdsDataAccess returns data in CSV lists

- All programs return "0" on Ok or an error number on error.

- All error messages appear on "stderr"

BDS Data Converter programming

- BDS Seismic data converters are implemented in the bdsDataLib as classes derived from BDS::DataFile. See BdsDataFile.h (Currently static library).

- This introduces a standard interface to all data format convertors.

- Add a new header and source file copied from BdsDataFile.* or another converter that is close.

- Modify create the code to read and/or write to the data format in question. The code also lists the formats it can handle.

- Add information on the new converter to BdsDataLib.cpp.

- Look at BdsDataFileBdrs.* example.

BDS Server Development

- The BdsServer is the heart of the system, be careful !

- Multi-threaded application with locking. Each clients connection runs as a separate thread.

- Control class is the heart of the program, implements all API calls.

- DataSet class handles seismic data access allowing a set of data from multiple files to be accessed and read.

- DataStore class handles actual access to the data store.

- When new API calls are added the BdsApi API glue code can be updated using the "make updateApi" command.

BdsServer: Structure
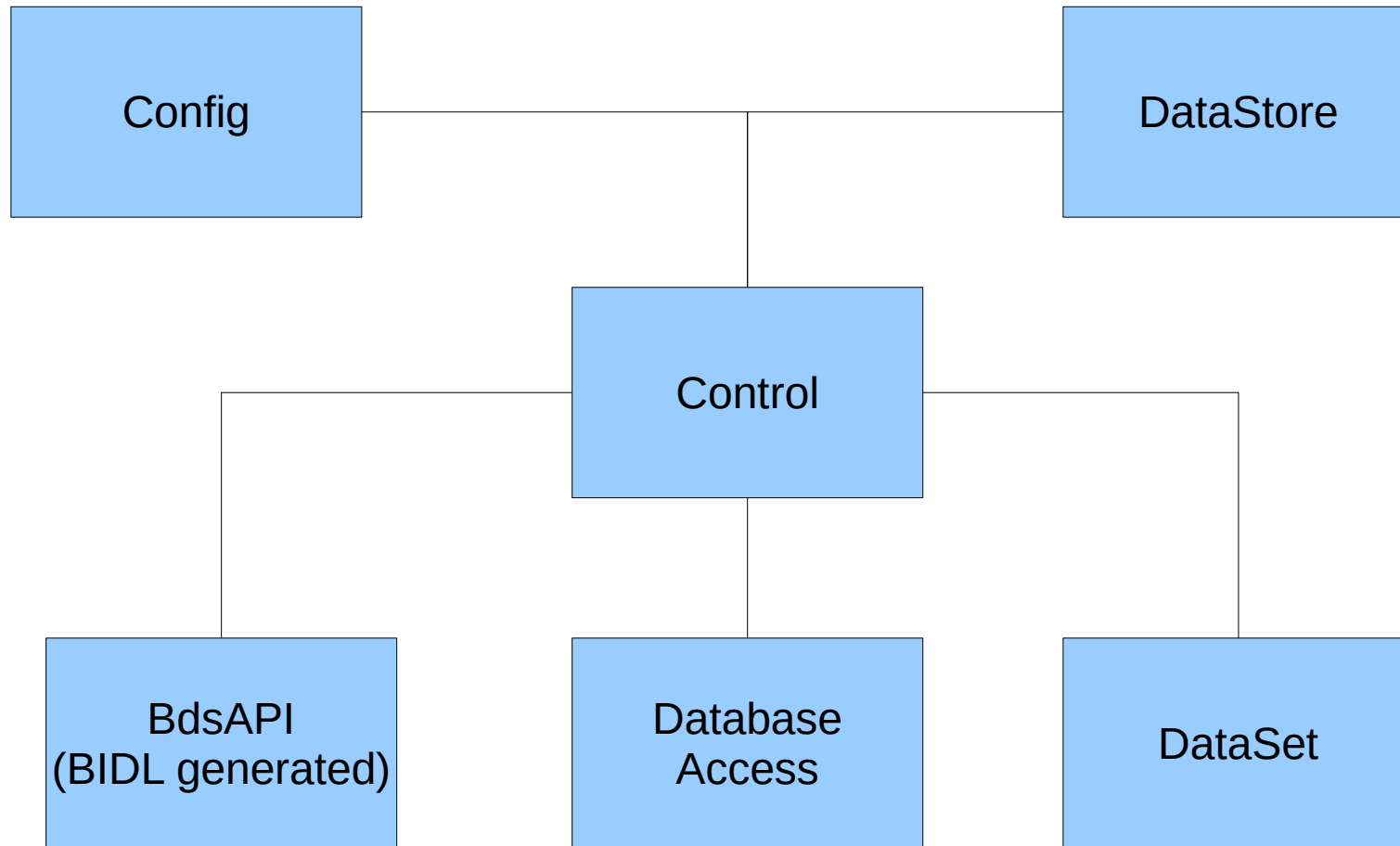
BDS API modifications

- The API is written using the BOAP IDL. This is in bdsLib.

- To modify or add new functionality, increment the apiVersion version parameter and make the changes.

- Run the "make all" command to rebuild the API library.

- Rebuild all BDS programs that use the API.

- Look at the current Bds.idl file.

- In BdsServer the "make updateApi" command can be issued to remake BdsApi.*

- The Control class will need the changed/additional functionality added.

BDS Database Changes

- MySQL database accessed solely by BdsServer.

- Only BdsServer and possibly BdsApi will need changing. Client programs unaffected.

- BdsSql directory has "bds.sql" defining the schema. "bdsData.sql" initial data entries.

- "bdsSqlUpdate-*.sql" this defines the database changes to the version number specified.

- Version number in "Config" table "SchemaVersion" value. Must match that in Makefile.config.

RPM packaging

- All code is packaged as binary and source RPMS.

- This allows the software to be easily updated on systems and modifications tracked.

- Allows dependencies to be managed.

- Allows for easy installation of complete systems with all necessary package lists.

- YUM archive at BEAM at:
  http://portal.beam.ltd.uk/dist/blacknest

- "yum update" will update all packages to current.

- "make rpms" will make the new BDS packages, version number is in Makefile.config

- bds.spec file may need modifications for new files etc

SVN Access

- All of the BDS source code is available, on-line, from the main SVN version system running at BEAM.

- To download the complete tree: "svn co https://portal.beam.ltd.uk/svn/blacknest/bds/trunk bds"

  "svn co https://portal.beam.ltd.uk/svn/blacknest/tapeDigitiser/trunk/libbeamdsp bds/libbeamdsp"

  "svn co https://portal.beam.ltd.uk/svn/blacknest/tapeDigitiser/trunk/libTapeDigitiser bds/libTapeDigitiser"

- To update tree: "svn update"

- Writing (committing) to the SVN tree requires a Blacknest SVN UserId and Password.

- To commit changes to BEAM's SVN repository use the command: "svn commit"

Further Help

- BDS Documentation site

- C++ programming books/course

- BDS is still in its early stages, a lot can change ...

- Bugs/Todo database at: https://portal.beam.ltd.uk/support/blacknest/info