BaseLib2 Tutorial Series
Configuration Database and Basic Type

André Neto

August, 2011

Outline

1. Introduction
2. Configuration Database
3. Basic Types

Data driven objects

- All BaseLib2 objects have an entry point for data driven configuration
- You should maximise the amount of data driven inputs to your code
  - Avoid hardcoded assumption
- A large portion of your code will be implemented here
  - Always perform checks sooner than later
  - Control boundaries
  - Allocate memory as soon as possible
    - Always check if memory was correcly allocated
- Log as much as you can

Configuration structure

- Configuration data is written using a syntax similar to C
- Data is organised in a tree with values assigned to keys
  - The value can be atomic (leaf)
  - or it can be a complex structure (node)

Example

```
+Control = {
    Class = ControlGAM
    Controller = {
        NoPlasmaVelocityGain = 0.0
        NoPlasmaCurrentGain = 40.0
        IPWaveform = {
            Times = {0 120}
            Amplitudes = {0.5 0.5}
            Rounding = 50
        }
...
```

Configuration database

- BaseLib2 provides a list of objects and functions to:
  - navigate on this tree
  - read values from it
  - write values to it
- This can be performed by any class implementing CDBVirtual
  - see ConfigurationDatabase.h and CDBVirtual.h in Level1
- The default CDBVirtual class used by ConfigurationDatabase is named CDB
  - see CDB.h in Level3

CDBVirtual.h

- Any CDBVirtual can be populated by reading a configuration stream (this is usually a file or a tcp stream)

Read and write stream

```
bool ReadFromStream(StreamInterface &s,StreamInterface *e=NULL,SortFilterFn *s=NULL);
bool WriteToStream(StreamInterface &s,StreamInterface *e=NULL,CDBWriteMode m=CDBWM_Tree);
```

Navigate

```
bool Move(const char *subTreeName);
bool MoveToChildren(int childNumber=0);
bool MoveToFather(int steps = 1);
bool AddChildAndMove(const char *subTreeName,SortFilterFn *s=NULL);
int NumberOfChildren();
```

Read and write data

```
bool GetArrayDims(int *size,int &maxDim,const char *configName,CDBArrayIndexingMode cdbaim =
CDBAIM_Flexible);
bool ReadArray (void *array,const CDBTYPE &valueType,const int *size,int nDim,const char
*configName);
bool WriteArray(const void *array,const CDBTYPE &valueType,const int *size,int nDim,const char
*configName,SortFilterFn *sorter=NULL);
```

Reading and writing data syntax

```
bool ReadArray (void *array,const CDBTYPE &valueType,const
int *size,int nDim,const char *configName);
```

- *array is a pointer to the place where data will be stored
- valueType is the type of data to be read (int, float, FString)
    - See CDBTypes.h in Level1
- nDim is the array dimension (0 for scalar, 1 for vector, 2 for matrix, ...)
- size is a pointer to an array with nDim entries, each with the size of each direction
    - scalar: nDim = 0 and size = NULL
    - vector: nDim = 1 and size[0] is the number of elements
    - matrix: nDim = 2 and size[0] is the number of rows and size[1] the number of vectors
- configName is the parameter name

Reading and writing data syntax

- Get the dimensions and the number of elements using:

```
bool GetArrayDims(int *size,int &maxDim,const char *configName,CDBArrayIndexingMode cdbaim =
CDBAIM_Flexible)
```

- *size is a pointer to the place where all sizes will be stored
- maxDim is the number of dimensions to be tested (number of elements in size)
- configName is the parameter name

ConfigurationDataBase Example (BaseLib2/Documentation/Tutorials/examples/CDBExample1.cpp)

Example code

```
//Configuration stored in an FString (usually this is a file or a tcp stream)
FString cdbTxt =
    "MySimpleClass = {\n"
    "    MyInt = 10\n"
    "    MyFloat = 5.0\n"
    "    MyString = \"A string\"\n"
    "    MyFloatArray = {-1.234 1.789 0.1233 1e2}\n"
    "    MyStringArray = {\"AAA\" \"BBB\" \"CCC\" ABC}\n"
    "    MyFloatMatrix = {\n"
    "        0 = {0.123 -1e3}\n"
    "        1 = {12345 .233}\n"
    "        2 = {-1 1.32}\n"
    "    }\n"
    "}\n";
```

ConfigurationDataBase Example
(BaseLib2/Documentation/Tutorials/examples/CDBExample1.cpp)

Example code

```
OBJECT_DLL(SimpleClass)
class SimpleClass :  public GarbageCollectable, public Object{
OBJECT_DLL_STUFF(SimpleClass)
private:
    int32 myInt;
    float myFloat;
    FString myString;
    float *myFloatArray;
    FString *myStringArray;
    float *myFloatMatrix;
/**
 * Configure an object using a configuration database
 */
bool ObjectLoadSetup(ConfigurationDataBase &cdb,StreamInterface *err){
    //Move to the place in the cdb
    if(!cdb->Move("MySimpleClass")){
        AssertErrorCondition(FatalError, "Could not move to
MySimpleClass");
        return False;
    }
```

ConfigurationDataBase Example (BaseLib2/Documentation/Tutorials/examples/CDBExample1.cpp)

Example code

```
//Read the int value.  Notice that scalar values can be read by passing NULL and 0
//for the size and dim of the array
if(!cdb->ReadArray(&myInt, CDBTYPE_int32, NULL, 0, "MyInt")){
    AssertErrorCondition(Warning, "MyInt was not defined.  Using default of %d", myInt);
}
if(!cdb->ReadArray(&myFloat, CDBTYPE_float, NULL, 0, "MyFloat")){
    AssertErrorCondition(Warning, "MyFloat was not defined.  Using default of %f", myFloat);
}
if(!cdb->ReadArray(&myString, CDBTYPE_FString, NULL, 0, "MyString")){
    AssertErrorCondition(Warning, "MyString was not defined.  Using default of %s",
myString.Buffer());
}
```

ConfigurationDataBase Example (BaseLib2/Documentation/Tutorials/examples/CDBExample1.cpp)

Example code

```
//In order to read a proper array first get the array dimensions (1=vector, 2=matrix, ...)  and
the size
//of each dimension
//Notice that the initialisation value of arrayDimension is also the maximum number of
dimensions searched
//by the ReadArray function
int32 arrayDimension = 2;
//Just put 2 to recycle later for matrix.  1 would be enough for vector
int32 arraySize[2];
if(cdb->GetArrayDims(arraySize, arrayDimension, "MyFloatArray")){
    if(arrayDimension == 1){
        AssertErrorCondition(Information, "MyFloatArray dimension is 1 as expected");
        AssertErrorCondition(Information, "MyFloatArray has %d elements", arraySize[0]);
        //Try to allocate memory
        myFloatArray = (float *)malloc(arraySize[0] * sizeof(float));
        if(myFloatArray == NULL){
            AssertErrorCondition(FatalError, "Failed to allocate %d bytes for myFloatArray",
(arraySize[0] * sizeof(float)));
            return False;
        }
        //Do the actual reading of values
        if(!cdb->ReadArray(myFloatArray, CDBTYPE_float, arraySize, arrayDimension,
"MyFloatArray")){
            AssertErrorCondition(FatalError, "Failed reading data to MyFloatArray");
            return False;
        }
```

CDBExtended

- The CDBExtended class provides an easier way of reading/writing values
  - Provides a dedicated function for each data type
  - Specify default values
  - See CDBExtended.h in Level2

Example functions are

```
bool ReadInt32(int32 &value,const char *configName,int32 defaultValue = 0);
bool ReadFloat(float &value,const char *configName,float defaultValue = 0);
bool ReadInt32Array(int32 *value,int *size,int nDim,const char *configName);
bool ReadFloatArray(float *value,int *size,int nDim,const char *configName);
```

CDBExtended example (BaseLib2/Documentation/Tutorials/examples/CDBExample2.cpp)

Example code

```
...
if(!cdbe.ReadInt32(myInt, "MyInt", 1234)){
...
if(!cdbe.ReadFloat(myFloat, "MyFloat", 1.234)){
...
int32 arrayDimension = 2;
int32 arraySize[2];
if(cdbe->GetArrayDims(arraySize, arrayDimension, "MyFloatArray")){
...
if(!cdbe.ReadFloatArray(myFloatArray, arraySize, arrayDimension,
"MyFloatArray")){
...
```

GlobalObjectDataBase

- Central pillar of any BaseLib2 application
  - Unique instance for each application (see GlobalObjectDataBase.h in Level1)
- Contains a reference to all the instantiated objects
  - It is a GCReferenceContainer
    - You can use all the GCReferenceContainer functions (Size(), Find(), ...)
- Objects are automatically created by providing a ConfigurationDataBase
  - All nodes starting with a + and with a **Class** = *className* attribute will be automatically created
    - className can also take the shared object library name where the class exists
    - The syntax is *sharedLibraryName::className* (e.g. `Class=MySharedLibrary::MyClass`)

GlobalObjectDataBase

- All objects that are automatically created, have their ObjectLoadSetup called
- Objects inherinting from GCNamedObject have their name automatically set as
  - The string between the + and the = (e.g. `+MyObj={`, would be named MyObj)...
  - ...or the value of a `Name=` parameter (if set)
- GCReferenceContainers and objects inherinting from GCReferenceContainer have their children automatically created

GlobalObjectDataBase example
(BaseLib2/Documentation/Tutorials/examples/CDBExample3.cpp)

Example code

```
...
//Inherit from GCNamedObject in order to automatically retrieve the name
OBJECT_DLL(SimpleClass)
class SimpleClass :  public GCNamedObject{
...
bool ObjectLoadSetup(ConfigurationDataBase &cdb,StreamInterface *err){
//Automatically read the object name
GCNamedObject::ObjectLoadSetup(cdb, err);
...
//Configuration stored in an FString (usually this is a file or a tcp
stream)
//Notice the + and the Class =
FString cdbTxt =
"+MySimpleClass1 = {\n"
"    Class = SimpleClass\n"
"    MyInt = 10\n"
...
"    }\n"
"}\n";
```

GlobalObjectDataBase example (BaseLib2/Documentation/Tutorials/examples/GODBExample1.cpp)

Example code

```
...
//Inherit from GCNamedObject in order to automatically retrieve the name
OBJECT_DLL(SimpleClass)
class SimpleClass :  public GCNamedObject{
...
bool ObjectLoadSetup(ConfigurationDataBase &cdb,StreamInterface *err){
//Automatically read the object name
GCNamedObject::ObjectLoadSetup(cdb, err);
...
//Configuration stored in an FString (usually this is a file or a tcp
stream)
//Notice the + and the Class =
FString cdbTxt =
"+MySimpleClass1 = {\n"
"    Class = SimpleClass\n"
"    MyInt = 10\n"
...
"    }\n"
"}\n";
```

GlobalObjectDataBase example (BaseLib2/Documentation/Tutorials/examples/GODBExample1.cpp)

Example code

```
//Create the configuration database and load from a string
ConfigurationDataBase cdb;
if(!cdb->ReadFromStream(cdbTxt)){
    CStaticAssertErrorCondition(FatalError, "Failed reading from stream!");
    return -1;
}
//Let the GlobalObjectDataBase automatically create the objects
if(!GetGlobalObjectDataBase()->ObjectLoadSetup(cdb, NULL)){
    CStaticAssertErrorCondition(FatalError, "Failed to load cdb");
    return -1;
}
//We can now look for the MyGCRef in the GlobalObjectDataBase
GCRTemplate<GCReferenceContainer> ref = GetGlobalObjectDataBase()->Find("MyGCRefC");
if(ref.IsValid()){
    CStaticAssertErrorCondition(Information, "Found MyGCRefC as expected");
    CStaticAssertErrorCondition(Information, "Number of children inside MyGCRefC is:  %d",
ref->Size());
}
else{
    CStaticAssertErrorCondition(FatalError, "Could not find MyGCRefC");
}
//List all the objects
GCRCLister lister;
GetGlobalObjectDataBase()->Iterate(&lister,GCFT_Recurse);
```

Basic types definition

- Provide unified way of describing a data type
  - Signed, unsigned
  - Floating point, integer
  - Number of bits
  - Most common data types already defined
    - BTDInteger, BTDFloat, ...
    - See BasicTypes.h
- Very useful to do arithmetic which depend on the type size
- Automatically converts data types in all the supported platforms
  - Very useful when you don't know the input type but want to have a generic function (see BTConvert)

Basic types example (BaseLib2/Documentation/Tutorials/examples/BTDExample1.cpp)

Example code

```
//Print information about a BasicTypeDescriptor
void PrintBTDInfo(BasicTypeDescriptor &btd){
    FString name;
    BTConvertToString(btd, name);
    CStaticAssertErrorCondition(Information, "Number of bytes in %s is %d (%d)", name.Buffer(),
btd.ByteSize(), btd.BitSize());
}

...
//Print information regarding a 32 bit integer
BasicTypeDescriptor anInt32 = BTDInt32;
PrintBTDInfo(anInt32);
//BasicTypeDescriptors can also be created from a string (for instance in a cfg file!)
BasicTypeDescriptor btdFloat;
BTConvertFromString(btdFloat, "float");
PrintBTDInfo(btdFloat);
//Convert between two types
int32 in = 123456789;
double out;
//Very useful if you don’t know the input type and want to convert to something using a generic
call
BTConvert(1, BTDDouble, &out, BTDInt32, &in);
CStaticAssertErrorCondition(Information, "Converted %d to %lf", in, out);
```

Training ideas

1. Design a PID object which allows the gains to be configured
2. Design a coil model object that allows to configure a coil inductance and resistance
3. Write an application which receives a configuration file as input and automatically creates and configures several PID and Model objects
    1. You can also use GCReferenceContainers to organise and make collections of different PID objects
4. After loading the application, list all the objects and print their main configuration values