Two simple databases already exists in BRAT.

1. Taking case of detector parameters (mainly simulation)
2. Geometry handling (from GBRAHMS output)

The databases are each just a simple flat ascii file. The DB manager reads data from the files when a new object is requested. They have been developed by (FV,KH) and used for simulation and MDC2.

- When a module request information the DB manager gets the information on behalf of the user module.
- The database information is kept in an object; the analysis job can also get a pointer to the object and call a Set method. This approach ensure all modules in the same task that uses this information will do this right. Particular important for Geometry information which may well be used by many modules, and a job might want to change information.
- An object is defined by two attributes only i.e.
  - Class of object e.g. BrDetectorParamsTPC
  - The name of the detector e.g. **T1**
  - Thus at present it does only allow for different choices of data sets through the choice of database i.e. **filename** which must be same for all object of same base class with a given analysis.
- Is this usable? I will let you be the judge; For now a suggestion it to extend it in a better way to be used for geometry and simulation parameters at least. Eventually the underlying access could/should be a (real) database, but the main idea is that user code is **NOT** changed. (except for a needed better split of simulation and detector physical parameters.

The Database Objects

The following is an extract of the parameter definition

class **BrDetectorParamsTPC** : public **BrDetectorParamsBase** {

```
public:
  BrDetectorParamsTPC();
  BrDetectorParamsTPC(Char_t *name,Char_t *title);
  BrDetectorParamsTPC(Char_t *name,Char_t *title,Char_t *filename);

  virtual ~BrDetectorParamsTPC();

  void ListParameters() const;
  void SetDefaultParams();
```

The user uses these method in the user module, as a guaranteed way to ensure consistency.

```
   Float_t GetTwoPar() {return fTwoPar;}
   Float_t GetDlong()  {return fDlong; }
   Float_t GetDtrans() {return fDtrans;}
….
void SetRowPosition(Int_t irow,Float_t xpos);

   virtual void SetASCIIParameters(Char_t *line);
```

Standard method (possible) called via the DbManager.

```
  private:
    void SetActiveParameters();
private:
  //
  // Simulation parameters
  //
  Float_t fTwoPar;      // Twoparticle resolution (not used)
  Float_t fDlong;         // Longitudinal Diffusion coeffecient
  Float_t fDtrans;       // Transverse   Diffisuion coefficient
```

```
Float_t fEff;                    // Efficiency for padrows
Float_t fAnodeGap;          // Distance between Pad and anode wire
                  // The proper sim and analysis parameter is really
                  // the pad resolution function (PRF)
//
// Gas and readout parameters
//
Float_t fDriftv;        // Drift velocity
Float_t fTimeBucket;       // Bucket size
```

Usage in the User Module

Here is the corresponding example for the TPC digitization routine how it setup the proper information using the detector name and class name.

The constructor to be used.

```
//_____________________________________________________________
_____________________
BrDigitizeTPC::BrDigitizeTPC(Text_t *Name,Char_t *Title) :
BrModule(Name, Title)
{
  fParams_p = 0;
  fVolumeParams_p = 0;
  fDetectorNode = 0;

//Get volume data with GeometryDbManager
  BrGeometryDbManager *gGeomDb =
BrGeometryDbManager::Instance();
  fVolumeParams_p = (BrDetectorVolume*)
    gGeomDb->GetDetectorVolume("BrDetectorVolume", Name);

  //Get parameters with ParameterDbManager
  BrParameterDbManager *gParamDb =
BrParameterDbManager::Instance();
  fParams_p = (BrDetectorParamsTPC*)
    gParamDb-
>GetDetectorParameters("BrDetectorParamsTPC",GetName());

  fAdcCut=1.0;  // some internal parameter
}
```

Usage of parameter in code.

```
cm_to_timeb = (Float_t)1.0/(fPparams_p->GetDriftv() * time_bucket);
  factor_to_channels = fParams_p->GetADCGain() * fParams_p-
>GetADCChannels() / (Float_t)1024.;
  //
```

Initialize the Database via the Database manager.

```
//
// Define detector volumes and parameters
// Creat an instance of the DbManager and declare the
// filename that digitization/reco routines will use.
// Do this before the constructors are called.
//

  BrGeometryDbManager *gGeomDb =
BrGeometryDbManager::Instance();
  gGeomDb->SetDebugLevel(0);
  gGeomDb->SetDbFileName(geo_fn);
  gGeomDb->SetDbMagnetFileName(mag_fn);

  BrParameterDbManager *gParamDb =
BrParameterDbManager::Instance();
  gParamDb->SetDebugLevel(0);
  gParamDb->SetDbParameterFileName("DetectorParameters.txt");
```

- The singleton approach means that a default DB manager will be create in case a usermodule requests access to it, but has not been setup in the main code.
- The Database manager owns the objects that (later) are requested and read into the code. User module only owns a pointer to the parameter objects.

Access and change a database parameter in the main program.

The code piece below instantiates the detector parameter object for the TPC and modifies the value. This value will be used correctly late when the user-module for the TPC digitization is instantiated.

```
BrDetectorParamsTPC * tpc_param =
     static_cast<BrDetectorParamsTPC*>
        (gParamDb-
>GetDetectorParameters("BrDetectorParamsTPC","T1"));
   tpc_param->SetADCGain(4.0);
   tpc_param =
    (BrDetectorParamsTPC*)
     gParamDb->GetDetectorParameters("BrDetectorParamsTPC","T2");
   tpc_param->SetADCGain(4.0);
```

Define the tpc1 digitization modules

```
BrDigitizeTPC *tpc1_digitize = new BrDigitizeTPC("T1","T1 Digitize
Module");
```

The module internally setups a call to the DB-manager, but need not to know anything about the actual file selected.