Hall A Analyzer Databases

- Mapping
  - Flat text files ("cratemap", "detector map")
- Geometry, Calibration, Configuration
  - Podd: Flat text files (key/value pairs with validity time-stamps)
- Future: Podd will allow user-selectable DB backends, *e.g.*
  - Hall A-style flat text files
  - Hall C-style parameter files
  - SQL server
  - CCDB (Hall B/D)

Example Hall A DB File

```
[ 2015-02-01 14:30:00 ]
#-- Mapping
B.mwdc.planeconfig = u1 u1p x1 x1p v1 v1p
                     u2 x2 v2
                     u3 u3p x3 x3p v3 v3p

B.mwdc.cratemap =  3 6  21 1877 500 96
                   4 4  11 1877 500 96
                   4 17 24 1877 500 96

#-- Geometry
B.mwdc.nwires       = 200  # Default
B.mwdc.u1.nwires    = 141  # Fewer wires

B.mwdc.size         = 2.0  0.5  0.0
B.mwdc.x1.size      = 1.4  0.35 0.0

#-- Configuration
B.mwdc.u.maxmiss    = 5

#-- Calibrations
B.mwdc.x1.res       = 0.255

[ 2015-02-02 16:45:00 ]
# only changed parameters here ...
B.mwdc.x1.res       = 0.258
```

Hall A Database API

- Retain 1.5 API in v1.6+ for backward compatibility
- Only minimal code changes required (see code snippets)
- v1.6+ API allows different backends, *e.g.*
    - Hall A-style flat files
    - Hall C-style parameter file
    - MySQL server
    - CDDB
    - others . . .
- Backend can be set and/or configured from replay script

Podd 1.5 Database Access

```
UserDetector::ReadDatabase( const TDatime& date ) {
  FILE* file = OpenFile( date );
  DBRequest request[] = {
    { "planeconfig", &planeconfig, kString },
    { "MCdata",      &mc_data,     kInt,     0, 1 },
    { 0 }
  };
  Int_t err = LoadDB( file, date, request, fPrefix );
  fclose(file);
};
```

Podd 1.6+ Database Access

```
THaInterface.C:
  THaDB* gHaDB = new THaFileDB( DB_DIR );  // Default DB

UserDetector::ReadDatabase( const TTimeStamp& date ) {
  DBRequest request[] = {
    { "planeconfig", &planeconfig, kString },
    { "MCdata",      &mc_data,     kInt,     0, 1 },
    { 0 }
  };
  Int_t err = LoadDB( date, request, fPrefix );
};
```

Database Support in Hall D JANA Framework I

- Geometry (“JGeometry”, “DGeometry”)
  - All geometry information accessed through JGeometry abstract base class. This is little more than a basic string-based database API.
  - JGeometry *constructor* takes run number argument for indexing.
  - Actual JGeometry object is re-instantiated for every run number.
  - Only XML backend implemented, although other backends conceivable. Uses Apache Xerces XML parser.
  - XML backend implements a simple caching mechanism for scalar items only
  - HDGEOMETRY library provides DGeometry wrapper, which offers numerous convenience methods that retrieve hardcoded keys and parse the corresponding values.
  - Individual detectors implement their own geometry data objects (JObject) and corresponding factories. These contain mostly hardcoded parameters. Some retrieve *some* parameters via DGeometry
  - No facilities exist to save, restore, stream or otherwise move any geometry objects to/from memory.

Hall D Geometry Database API Snippets

JGeometry.h

```
class JGeometry{
  public:
  JGeometry(string url, int run, string context="default");
  ...
  // Virtual methods called through base class
  virtual bool Get(string xpath, string &sval)=0;
  virtual bool Get(string xpath, map<string, string> &svals)=0;
  virtual bool GetMultiple(string xpath, vector<string> &vsval)=0;
  virtual bool GetMultiple(string xpath, vector<map<string, string> >&vsvals)=0;
  ...
};
```

DGeometry.h

```
class DGeometry{
  public:
  DGeometry(JGeometry *jgeom, DApplication *dapp, unsigned int runnumber);
  ...
  // Convenience methods
  bool GetFDCWires(vector<vector<DFDCWire *> >&fdcwires) const;
  bool GetFDCCathodes(vector<vector<DFDCCathode *> >&fdccathodes) const;
  bool GetFDCZ(vector<double> &z_wires) const; ///< z-locations for each of the FDC wire planes in cm
  bool GetFDCStereo(vector<double> &stereo_angles) const; ///< stereo angles of each of the FDC wire layers
  bool GetFDCRmin(vector<double> &rmin_packages) const; ///< beam hole size for each FDC package in cm
  bool GetFDCRmax(double &rmax_active_fdc) const; ///< outer radius of FDC active area in cm
  ...
};
```

Database Support in Hall D JANA Framework II

- Calibration (“`JCalibration`”)
  - All calibration accessed via `JCalibration` abstract base class. Similar to `JGeometry`, but additionally
    - ★ Provides both read and write functions
    - ★ Records all access requests
    - ★ Implements methods to write all actually used keys (but not values) to file
  - *Constructor* takes run number argument for indexing. Actual calibration class is re-instantiated for every run number.
  - CCDB backend available. Implements only `GetCalib` functions, not `PutCalib`, i.e. it's a read-only API.
  - `JCalibrationFile` backend supports reading and writing to a group of files on local disk.

Hall D Calibration Database API Snippets

JCalibration.h

```
class JCalibration{
  public:
  JCalibration(string url, int run, string context="default");
  ...
  // Returns "false" on success and "true" on error
  virtual bool GetCalib(string namepath, map<string, string> &svals, int event_number=0)=0;
  virtual bool GetCalib(string namepath, vector< map<string, string> > &svals, int event_number=0)=0;

  template<class T> bool Get(string namepath, map<string,T> &vals, int event_number=0);
  template<class T> bool Get(string namepath, vector<T> &vals, int event_number=0);
  template<class T> bool Get(string namepath, vector< map<string,T> > &vals, int event_number=0);
  template<class T> bool Get(string namepath, vector< vector<T> > &vals, int event_number=0);
  ...
  virtual bool PutCalib(string namepath, int run_min, int run_max, int event_min, int event_max,
    string &author, map<string, string> &svals, string comment="");
  ...
    template<class T>bool Put(string namepath, int run_min, int run_max, int event_min, int event_max,
      string &author, map<string,T> &vals, const string &comment="");
};

template<class T>
bool JCalibration::Get(string namepath, map<string,T> &vals, int event_number) {
  // Get values in the form of strings
  map<string, string> svals;
  bool res = GetCalib(namepath, svals, event_number);
  RecordRequest(namepath, typeid(map<string,T>).name());
  // Parse string to T
  ...
  return res;
}
```

Database Support in Hall D JANA Framework III

- Configuration ("`JParameter`")
  - Implemented via `JParameterManager` API
  - `JParameterManager` is a thread-safe singleton
  - Essentially a map of keys to scalar values (but implemented as a `vector` . . .)
  - All values stored as strings internally. Conversion to numerical data types provided via stringstream.
  - Provides methods for writing and reading to/from file
- Resource Manager ("`JResourceManager`")
  - (not yet studied . . .)
- Mapping ("`DTranslationTable`")
  - Support actually not implemented in JANA, but in Hall D-specific library (`TTAB` module). `TTAB` is compiled into the EVIO decoder.
  - XML format, very Hall D-specific
  - All Hall D detector classes are compiled in. Detector indexing schemes are hardcoded in `DTranslationTable.h`.
  - XML file may be stored in CCDB (as one large string!)
  - Uses `libexpat` for XML parsing