https://confluence.slac.stanford.edu/display/SAS/Code+Reviews

/nfs/slac/g/glast/users/glground/heather/ft2UtilCodeReview/ft2UtilBuild

Major Comments:

In long, important functions, especially Livetime::operator()(...), it is a good idea to refactor the code blocks into member functions (see comment on Livetime class, below). This makes the outer calling function easier to read; and if you make the interior member functions part of the public interface, you can write test code that exercises each individual method (or you can specify the testing class as a friend class (not my preference)).

The package should include small Magic7, digi, and FT2 files for testing purposes, and the test code should exercise all of the corner cases that are expressed in the various if-else blocks.

I'd recommend moving ExtendedPointingInfo and Spacecraft to the astro pacakage since these are generally useful to both ST and GR. ExtendedPointingInfo could be made a true subclass of PointingInfo (with corresponding virtual functions) or the member functions could just be added to the existing PointingInfo class.

Minor comments:

One should define const member functions and pass values by reference whenever possible. Generally, data members should not be in the public part of the class interface, i.e., they should be encapsulated with access via const references. The use of include statements should be minimized in header files.

Class-specific comments:

* In usage example, since StatusFile::operator()(...) returns a Status object by value, it is more efficient to set it equal to a local instance of that class rather than forcing recomputation with a new call to operator()(...):

```
Status my_status = statusfile(time);
double RA_SCX = my_status.spacecraft.xAxis().ra();
double RA_LATX = my_status.lat.xAxis().ra();
```

* It is better to have the data members Status::spacecraft, Status::lat, Status::sun fully encapsulated and returned as const references via member functions:

```
class Status:

public:
  const ExtendedPointingInfo & spacecraft() const {
     return m_spacecraft;
  }
  const LatInfo & lat() const {
     return m_lat;
  }
  const astro::SkyDir & sun() const {
```

```
      return m_sun;
    }

  private:
    ExtendedPointingInfo m_spacecraft;
    LatInfo m_lat;
    astro::SkyDir m_sun;
  }
```

* StatusFile::startTimes:
  Change to private data member and expose via const reference function call.

* StatusFile::operator()(double):

  * Rename m_current_pointing to current_pointing. "m_"-prefix should be reserved for data members.

  * One should use a const reference instead, since that is what PointingHistory::operator()(double) returns:

    ```
    const astro::PointingInfo & current_pointing(m_pointingHistory(time));
    ```

  * Should also use const references for Quaternion objects, e.g.,

    ```
    const astro::Quaternion & highBound_quat(itor->second);
    ```

* ExtendedPointingInfo:

  * quaternion member function should be a const reference:

    ```
    const astro::Quaternion & quaternion() const {
       return m_quaternion;
    }
    ```

  * The m_position data member isn't needed.

* LatInfo:

  * quaternion() should be const. This is possible if ExtendedPointingInfo::quaternion() is also const.

  * mode(), config(), dataQuality() should all be const.

  * angles_have_been_updated and boresight should be static data members of the class, with appropriate interfaces:

    ```
    class LatInfo {
    public:

       static bool angles_have_been_updated() {
          return s_angles_have_been_updated;
       }

       static const Boresight & boresight() {
          return s_boresight;
       }
    ```

```
    private:

       static bool s_angles_have_been_updated;
       static Boresight s_boresight;
    };
```

  * getBoresightMatrix() should probably be a static member function.

* LatCondition:

  * mode, config, and dataQuality should be encapsulated and made
    accessible via const references returned by member functions.

* Boresight:
  * getQuaternion() should be const.
  * getRotation() should be const and return a const reference:

```
    const CLHEP::HepRotation & getRotation() const;
```

  * XmlParser is a private member function, so don't need to pass Rx, Ry, Rz
    as function arguments. Just assign directly:

```
    m_x = xmlBase::Dom::getDoubleAttribute(attElt, "Rx");
```

    Perhaps rename to xmlParser to disambiguate from xmlBase::XmlParser.

    Should rethrow the xmlBase::DomException's after catching and printing
    error messages.

* Configuration:

  * Should use more standard syntax for implementing the Singleton pattern:

```
  class Configuration {
  public:
    static Configuration & instance() {
       if (s_instance == 0) {
          s_instance = new Configuration();
       }
       return *s_instance;
    }
  }

  private:
    Configuration();
    static Configuration * s_instance;
  };
```

  * It would be good to add a member function to read in a
    user-specifed configuration file, and an option should be added to
    the makeFT2 application should allow the user to override from the
    default.

* Extrapolator, Interpolator:

  * evaluateIn(double) should be const (argument does not need to be const).

  * Delete #include "astro/PointingInfo.h" from header files.
  * Can use forward declarations of TGraph, TF1 in Extrapolator.h
  * Add #include "CLHEP/Vector/ThreeVector.h" to Interpolator.h

* OrbMessage:

  * In constructor, pass m7OrbMessageTokens by reference.
  * all member functions should be const.
  * getPosition and getVelocity should return const references.
  * Delete #include "astro::PointingInfo.h" in OrbMessage.h
  * Add #include "CLHEP/Vector/ThreeVector.h" in OrbMessage.h

* AttMessage:

  * In constructor, pass m7AttMessageTokens by reference.
  * all member functions should be const.
  * getQuaternion and getRotVelocity should return const references.
  * Delete #include "astro::PointingInfo.h" in AttMessage.h
  * Add #include "CLHEP/Vector/ThreeVector.h" in AttMessage.h
  * Add #include "astro/Quaternion.h" in AttMessage.h

* Magic7:

  * timeIntervals should be encapsulated (and renamed m_timeIntervals)
  * Data members should be called m_attMessages, m_orbMessages.

  * getQuaternion(...), normalize(...), getPosition(...), getMode(...),
    getInSAA(...) should be const.

  * The constructor and member functions are pretty complicated.
    Is there test code available that exercises all of the special cases?
    Can we put an example magic7 file in the data subdirectory?

* Livetime:

  * Delete #include "astro/PointingInfo.h" from Livetime.h
  * Use forward declarations for TFile, TTree, DigiEvent.
  * Pass arguments to constructor as const std::string & .
  * Use static_cast<int>(runID) instead of (int)runID.
  * operator()(double, double) is very long. It would be useful to refactor
    the internal implementation into smaller member function calls with
    descriptive names like "handleReconCrashes()".
  * An example input file and test code should be made available.