{{lastupdated_at}} by {{lastupdated_by}}

GTime

The GTime class is intended to handle times in an abstract way.

Existing implementation

The existing implementation tries to incorporate different time systems in a single class. The implementation is definitely not satisfactory, but it satisfies the current needs (which are not high as no time conversion is in fact used). In the long run, the class should be redefined to handle various time systems efficiently (see Feature #284).

Possible new implementation

Here some thoughts about a possible new implementation.

The GTime class can be the base class for instrument specific time classes. One option is that GTime is not abstract, i.e. it can be used without any instrument information. The class has just a single member, which is the time in seconds in UTC with a well defined zero point (for example January 1st, 2000, this is to be defined). The class has then basic access methods that allow setting and returning of the time in seconds, days, MJD and JD. Maybe also some string formating should be supported.

Instrument specific time classes are then derived from the GTime class and implement conversion classes from the instrument specific time to UTC. The conversion can be done via constructors or specific conversion methods. It remains to be seen whether these conversion constructors and methods exist only for the derived classes, or whether they overload base class methods.

Alternatively, the instrument specific classes could store the time in the instrument time reference, and then implement the proper conversion methods to go to UTC. This avoids time conversion when they are not necessary (in most cases). Time conversion would thus only exist on an instrument specific level. The GTime could then even be abstract, and a GUTCTime class could be added to handle general times in the UTC time reference.

When deciding which method is better we should think about time usage in GGti. GGti is a general class, but it should also handle instrument specific times. There a several solutions to this:

- Use GTime for storing times in instrument specific reference. This is more a kluge that works if the GGti is only used within the instrument specific frame, which we can never be sure, however. The advantage is that there is little overhead, the disadvantage is that we never can be sure what time reference is in the GTime. This is prone to errors.
- Use an instrument specific callback when reading the time in GGti. This looks complicated and implies a bunch of time conversions.
- Use a registry for the instrument specific times, and rely on information in the GTI header to allocate the correct instrument specific time class. The drawback is that it does not work on FITS files with incomplete header information (i.e. those not respecting the OGIP format).

At the end it seems that the abstract solution with an instrument specific registry seems to be the best. Each time can then say in which reference it is, and we can fall back to GUTCTime if we need an instrument independent time.

The abstract GTime class would then have the pure virtual methods:

```
virtual double jd(void) const = 0;
virtual double mjd(void) const = 0;
virtual double secs(void) const = 0;
virtual double days(void) const = 0;
virtual void   jd(const double& time) = 0;
virtual void   mjd(const double& time) = 0;
virtual void   secs(const double& time) = 0;
virtual void   days(const double& time) = 0;
```

for reading and setting the times. The jd and mjd methods perform time conversion, the secs and days return the time in native format.

But what is the real advantage of having instrument specific times? Can't we just convert always to the same time reference, and

then just go with this?

I'm no longer convinced that the abstract base class scheme is really what we want. One reason is that time manipulation becomes more complicated. For example, we can't simply write

```
GTime tstart;
GTime tstop;
...
GTime time = tstop - tstart;
```

i.e. simple operations become impossible (as GTime is abstract). We then would need to implement this operator for all classes, e.g.

```
GCTATime tstart;
GCTATime tstop;
...
GUTCTime time = tstop - tstart;
```

In fact, the only advantage of having derived classes is that the reference time can be stored statically, and also complicated time transformations can be coded. This would allow handling for example the specific onboard time formats for some satellites. But do we really need this for high-level analysis?

It seems better to keep the actual structure and to store the time reference information in the GGti class. We may even add a GTimeReference class for this, so that time references can be delt with in any context.

New implementation

The GTime holds the time in a GammaLib specific time reference. The time is stored in a double precision value in seconds.

The following methods are implemented to access or modify the time value:

```
double      jd(void) const;              //!< Return time in Julian Days
double      mjd(void) const;             //!< Return time in Modified Julian Days
double      secs(void) const;            //!< Return time in seconds (GammaLib specific reference time)
double      days(void) const;            //!< Return time in days (GammaLib specific reference time)
std::string utc(void) const;             //!< Return time as UTC string
void        jd(const double& time);      //!< Set time in Julian Days
void        mjd(const double& time);     //!< Set time in Modified Julian Days
void        secs(const double& time);    //!< Set time in seconds (GammaLib specific reference time)
void        days(const double& time);    //!< Set time in days (GammaLib specific reference time)
void        utc(const std::string& time); //!< Set time from UTC string
```

The time reference is implemented by a GTimeReference class. This class should provide methods to transform the time in a given reference to the GammaLib specific reference time. The class should be able to read (write) the time reference information from (to) an OGIP FITS HDU.

The time reference can be stored in the GGti class, allowing the automatic conversion of the time values into the GammaLib specific time reference. Keeping this information in the GGti class allows then writing back the time in the specified time reference.

Time systems

Time Systems in a Nutshell

The Systeme International (SI) second is defined as the duration of 9,192,631,770 cycles of radiation corresponding to the transition between two hyperfine levels of the ground state of cesium 133.

Universal Time 1 (UT1) is the time system based on the rotation of the earth. Because of changes in the earth's rotation rate, in UT1 a day is not exactly 86400 s.

Coordinated Universal Time (UTC) provides a uniform-rate time system referenced to atomic clocks where a day is 86400 s. To keep UT1 and UTC within 0.9 s, a leap second is added to UTC as needed, typically every few years (see the USNO leap second history). UTC is the same as Greenwich Mean Time (GMT) or Zulu time (for the military).

Leap seconds can cause errors in measurements that straddle the addition of a leap second. The Terrestrial Time (TT) is a uniform rate time system referenced to the geoid without leap seconds. Effectively, TT time is greater than UTC time by a number that increases by 1 second every time a leap second is added to UTC. Below a table of leap seconds for different periods in time:

Period	Leap seconds
1990 Jan. 1 - 1991 Jan. 1	25 s
1991 Jan. 1 - 1992 Jul. 1	26 s
1992 Jul. 1 - 1993 J	27 s

ul 1	
1 9 9 3 J ul . 1 - 1 9 9 4 J ul . 1	2 8 s
1 9 9 4 J ul . 1 - 1 9 9 6 J a n. 1	2 9 s
1 9 9 6 J a n. 1 - 1 9 9 7 J ul . 1	3 0 s
1 9 9 7 J ul . 1 - 1 9 9	3 1 s

9 Jan 1	
1999 Jan. 1 - 2006 Jan 1	32s
2006 Jan. 1 - 2009 Jan 1	33s
2009 Jan. 1 - 2012 Jul 1	34s
2012 Jul. 1 -	35s

Another method of avoiding leap seconds over a time span of a number of years is to use the number of seconds relative to a reference time. This is the method that astrophysical missions often use, where the number of seconds is called 'Mission Elapsed Time' (MET). Because MET and TT are both continuous uniform-rate time systems, MET and TT will always be offset from each other by a constant. Note that the time system should be included in specifying a reference time (e.g., whether the reference time is midnight on a particular date in the TT or UTC system).

The Global Positioning System (GPS) uses its own continuous uniform-rate time system that is related by a constant offset (13.184 s) to TT.

The Julian Date (JD) is the number of days since Greenwich mean noon on January 1, 4713 B.C.E. Since JD is a large number—midnight at the beginning of January 1, 2008, corresponds to JD=2454466.5—and our calendar uses midnight as the beginning and end of a calendar day, the Modified Julian Date (MJD) has been defined as MJD=JD-2400000.5. Midnight (i.e., 0h:0m:0s) differs between the UTC and TT systems, and therefore one should specify whether a JD or MJD date is in the UTC or TT system.

Below

Date (UTC)	**MJD (UTC)**	**MJD (TT)**	**TT-UTC**	**Notes**
January 1, 1998			63.184 sec	XMM-Newton reference
January 1, 2001 (UTC)	51910.0	51910.00074287037037703703	64.184 sec	Fermi-LAT reference
January 1, 2010 (UTC)	55197.0	55197.000766018518519	66.184 sec	**GammaLib reference**

Further readings:

- attachment:Circular_179.pdf
- Systems of Time
- Leap seconds

Files

Circular_179.pdf	1.57 MB	12/17/2012	Knödlseder Jürgen