```
/*
          time.h - low level time and date functions
        */


        /*
          July 3 2011 - fixed elapsedSecsThisWeek macro (thanks Vincent Valdy for this)
                      - fixed  daysToTime_t macro (thanks maniacbug)
        */


        #ifndef _Time_h
        #ifdef __cplusplus
        #define _Time_h


        #include <inttypes.h>
        #ifndef __AVR__
        #include <sys/types.h> // for __time_t_defined, but avr libc lacks sys/types.h
        #endif




        #if !defined(__time_t_defined) // avoid conflict with newlib or other posix libc
        typedef unsigned long time_t;
        #endif




        // This ugly hack allows us to define C++ overloaded functions, when included
        // from within an extern "C", as newlib's sys/stat.h does.  Actually it is
        // intended to include "time.h" from the C library (on ARM, but AVR does not
        // have that file at all).  On Mac and Windows, the compiler will find this
        // "Time.h" instead of the C library "time.h", so we may cause other weird
        // and unpredictable effects by conflicting with the C library header "time.h",
        // but at least this hack lets us define C++ functions as intended.  Hopefully
        // nothing too terrible will result from overriding the C library header?!
        extern "C++" {
        typedef enum {timeNotSet, timeNeedsSync, timeSet
        }  timeStatus_t ;


        typedef enum {
```

```
    dowInvalid, dowSunday, dowMonday, dowTuesday, dowWednesday, dowThursday,
dowFriday, dowSaturday
} timeDayOfWeek_t;


typedef enum {
    tmSecond, tmMinute, tmHour, tmWday, tmDay,tmMonth, tmYear, tmNbrFields
} tmByteFields;


typedef struct  {
  uint8_t Second;
  uint8_t Minute;
  uint8_t Hour;
  uint8_t Wday;   // day of week, sunday is day 1
  uint8_t Day;
  uint8_t Month;
  uint8_t Year;   // offset from 1970;
}       tmElements_t, TimeElements, *tmElementsPtr_t;


//convenience macros to convert to and from tm years
#define  tmYearToCalendar(Y) ((Y) + 1970)  // full four digit year
#define  CalendarYrToTm(Y)   ((Y) - 1970)
#define  tmYearToY2k(Y)      ((Y) - 30)    // offset is from 2000
#define  y2kYearToTm(Y)      ((Y) + 30)


typedef time_t(*getExternalTime)();
//typedef void  (*setExternalTime)(const time_t); // not used in this version




/*==============================================================================*
/
/* Useful Constants */
#define SECS_PER_MIN  ((time_t)(60UL))
#define SECS_PER_HOUR ((time_t)(3600UL))
#define SECS_PER_DAY  ((time_t)(SECS_PER_HOUR * 24UL))
#define DAYS_PER_WEEK ((time_t)(7UL))
#define SECS_PER_WEEK ((time_t)(SECS_PER_DAY * DAYS_PER_WEEK))
#define SECS_PER_YEAR ((time_t)(SECS_PER_DAY * 365UL)) // TODO: ought to handle
leap years
#define SECS_YR_2000  ((time_t)(946684800UL)) // the time at the start of y2k
```

```
/* Useful Macros for getting elapsed time */
#define numberOfSeconds(_time_) ((_time_) % SECS_PER_MIN)
#define numberOfMinutes(_time_) (((_time_) / SECS_PER_MIN) % SECS_PER_MIN)
#define numberOfHours(_time_) (((_time_) % SECS_PER_DAY) / SECS_PER_HOUR)
#define dayOfWeek(_time_) ((((_time_) / SECS_PER_DAY + 4)  % DAYS_PER_WEEK)+1) //
1 = Sunday
#define elapsedDays(_time_) ((_time_) / SECS_PER_DAY)  // this is number of days
since Jan 1 1970
#define elapsedSecsToday(_time_) ((_time_) % SECS_PER_DAY)   // the number of
seconds since last midnight
// The following macros are used in calculating alarms and assume the clock is
set to a date later than Jan 1 1971
// Always set the correct time before setting alarms
#define previousMidnight(_time_) (((_time_) / SECS_PER_DAY) * SECS_PER_DAY)  //
time at the start of the given day
#define nextMidnight(_time_) (previousMidnight(_time_)  + SECS_PER_DAY)   // time
at the end of the given day
#define elapsedSecsThisWeek(_time_) (elapsedSecsToday(_time_) +
((dayOfWeek(_time_)-1) * SECS_PER_DAY))   // note that week starts on day 1
#define previousSunday(_time_) ((_time_) - elapsedSecsThisWeek(_time_))      //
time at the start of the week for the given time
#define nextSunday(_time_) (previousSunday(_time_)+SECS_PER_WEEK)          //
time at the end of the week for the given time




/* Useful Macros for converting elapsed time to a time_t */
#define minutesToTime_t ((M)) ( (M) * SECS_PER_MIN)
#define hoursToTime_t   ((H)) ( (H) * SECS_PER_HOUR)
#define daysToTime_t    ((D)) ( (D) * SECS_PER_DAY) // fixed on Jul 22 2011
#define weeksToTime_t   ((W)) ( (W) * SECS_PER_WEEK)


/*==============================================================================*/
/*  time and date functions   */
int     hour();            // the hour now
int     hour(time_t t);    // the hour for the given time
int     hourFormat12();    // the hour now in 12 hour format
int     hourFormat12(time_t t); // the hour for the given time in 12 hour format
uint8_t isAM();            // returns true if time now is AM
uint8_t isAM(time_t t);    // returns true the given time is AM
uint8_t isPM();            // returns true if time now is PM
uint8_t isPM(time_t t);    // returns true the given time is PM
int     minute();          // the minute now
int     minute(time_t t);  // the minute for the given time
int     second();          // the second now
int     second(time_t t);  // the second for the given time
int     day();             // the day now
int     day(time_t t);     // the day for the given time
int     weekday();         // the weekday now (Sunday is day 1)
int     weekday(time_t t); // the weekday for the given time
int     month();           // the month now  (Jan is month 1)
int     month(time_t t);   // the month for the given time
int     year();            // the full four digit year: (2009, 2010 etc)
int     year(time_t t);    // the year for the given time


time_t now();              // return the current time as seconds since Jan 1 1970
void    setTime(time_t t);
void    setTime(int hr,int min,int sec,int day, int month, int yr);
void    adjustTime(long adjustment);


/* date strings */
#define dt_MAX_STRING_LEN 9 // length of longest date string (excluding
terminating null)
char* monthStr(uint8_t month);
char* dayStr(uint8_t day);
char* monthShortStr(uint8_t month);
char* dayShortStr(uint8_t day);

/* time sync functions       */
timeStatus_t timeStatus(); // indicates if time has been set and recently
synchronized
void    setSyncProvider( getExternalTime getTimeFunction); // identify the
external time provider
void    setSyncInterval(time_t interval); // set the number of seconds between
re-sync


/* low level functions to convert to and from system time                     */
void breakTime(time_t time, tmElements_t &tm);  // break time_t into elements
time_t makeTime(const tmElements_t &tm);  // convert time elements into time_t


} // extern "C++"
#endif // __cplusplus
#endif /* _Time_h */
```