Motivation

Since DataFrame is a statistical library, it very often deals with time-series data. So, it needs to keep track of time.
The most efficient way of indexing DataFrame by time is to use an index type of time_t for second precision or double or long long int for more precision. But DateTime class for if you need a more elaborate handling of time. Or if you need a general handy DateTime object.

Code structure

Both the header (DateTime.h) and source (DateTime.cc) files are part of the DataFrame project. They are in the usual include and src directories.

Build Instructions

Follow the DataFrame build instructions.

Example

This library could have up to Nano second precision depending on what systems calls are available.
These are example code:

```
DateTime    now;
DateTime    gmt_now (DT_TIME_ZONE::GMT);
DateTime    hk_now (DT_TIME_ZONE:: AS_HONG_KONG);

cout << "Local Time is: " << now.string_format (DT_FORMAT::DT_TM2) << std::endl;
cout << "GMT Time is: " << gmt_now.string_format (DT_FORMAT::DT_TM2) << std::endl;

double  diff = now.diff_seconds (gmt_now);

now = 19721202;
gmt_now = 19721210;
diff = now.diff_weekdays (gmt_now);

now.add_days(3)
now.add_weekdays(-2);
```

For more examples see file *date_time_tester.cc*

Types

```
enum class DT_FORMAT : unsigned short int  {
    AMR_DT = 1,                 // e.g. 09/16/99
    AMR_DT_CTY = 2,             // e.g. 09/16/1999
    EUR_DT = 3,                 // e.g. 16/09/99
    EUR_DT_CTY = 4,             // e.g. 16/09/1999
    DT_TM = 5,                  // e.g. 09/16/1999 13:51:04
    SCT_DT = 6,                 // e.g. Sep 16, 1999
    DT_MMDDYYYY = 7,            // e.g. 09161999
    DT_YYYYMMDD = 8,            // e.g. 19990916
    DLR_MNY = 9,                // e.g. $ 120350045
    DLR_MNY_C = 10,             // e.g. $ 120,350,045
    DLR_MNY_C_DM = 11     // e.g. $ 120,350,045.53
    VAL_32ND = 12,              // e.g. 105-164
    VAL_64TH = 13,              // e.g. 105=33
    DT_TM2 = 14,                 // e.g. 09/16/1999 13:51:04.256
    DT_DATETIME = 15,           // e.g. 20010103   09:31:15.124
    DT_FAME_DATE = 16,          // e.g. 27Sep2001
    DT_PRECISE = 17             // e.g. 1516179600.874123908 = Epoch.Nanoseconds
};
```

These constants are used for formatting date/time into strings.

```
enum class DT_TIME_ZONE : short int  {
    LOCAL = -2,
    GMT = 0,
    AM_BUENOS_AIRES = 1,
    AM_CHICAGO = 2,
    AM_LOS_ANGELES = 3,
    AM_MEXICO_CITY = 4,
    AM_NEW_YORK = 5,
    AS_DUBAI = 6,
    AS_HONG_KONG = 7,
    AS_SHANGHAI = 8,
    AS_SINGAPORE = 9,
    AS_TEHRAN = 10,
    AS_TEL_AVIV = 11,
    AS_TOKYO = 12,
    AU_MELBOURNE = 13,
    AU_SYDNEY = 14,
    BR_RIO_DE_JANEIRO = 15,
    EU_BERLIN = 16,
    EU_LONDON = 17,
    EU_MOSCOW = 18,
    EU_PARIS = 19,
    EU_ROME = 20,
    EU_VIENNA = 21,
    EU_ZURICH = 22,
```

```
    UTC = 23,
    AS_SEOUL = 24,
    AS_TAIPEI = 25,
    EU_STOCKHOLM = 26,
    NZ = 27,
    EU_OSLO = 28,
    EU_WARSAW = 29,
    EU_BUDAPEST = 30
};
```
These are the available time zones.

```
enum class DT_WEEKDAY : unsigned char  {
    BAD_DAY = 0,
    SUN = 1,
    MON = 2,
    TUE = 3,
    WED = 4,
    THU = 5,
    FRI = 6,
    SAT = 7
};
```
Week days: 1 - 7 (Sunday - Saturday)

```
enum class DT_MONTH : unsigned char  {
    BAD_MONTH = 0,
    JAN = 1,
    FEB = 2,
    MAR = 3,
    APR = 4,
    MAY = 5,
    JUN = 6,
    JUL = 7,
    AUG = 8,
    SEP = 9,
    OCT = 10,
    NOV = 11,
    DEC = 12
};
```
Months: 1 - 12 (January - December)

```
enum class DT_DATE_STYLE : unsigned char  {
    YYYYMMDD = 1,
    AME_STYLE = 2,
    EUR_STYLE = 3
};
```
These constants are used for parsing data
AME_STYLE: MM/DD/YYYY
EUR_STYLE: YYYY/MM/DD

DateType = unsigned int	YYYYMMDD
DatePartType = unsigned short int	year, month etc.
HourType = unsigned short int	0 - 23
MinuteType = unsigned short int	0 - 59
SecondType = unsigned short int	0 - 59
MillisecondType = short int	0 - 999
MicrosecondType = int	0 - 999,999
NanosecondType = int	0 - 999,999,999
EpochType = time_t	Signed epoch

Methods

explicit DateTime (DT_TIME_ZONE the_zone = DT_TIME_ZONE::LOCAL) noexcept;
A constructor that creates a DateTime initialized to now.

the_zone: Desired time zone from DT_TIME_ZONE above.

```
explicit DateTime (DateType d,
                   HourType hr = 0,
                   MinuteType mn = 0,
                   SecondType sc = 0,
                   NanosecondType ns = 0,
                   DT_TIME_ZONE tz = DT_TIME_ZONE::LOCAL) noexcept;
```
The constructor that creates a DateTime based on parameters passed.

d: Date e.g. 20180112
hr: Hour e.g. 13
mn: Minute e.g. 45
sc: Second e.g. 45
ns: Nano-second e.g. 123456789
the_zone: Desired time zone from DT_TIME_ZONE above.

```
explicit DateTime (const char *s,
                   DT_DATE_STYLE ds = DT_DATE_STYLE::YYYYMMDD,
                   DT_TIME_ZONE tz = DT_TIME_ZONE::LOCAL);
```
The constructor that creates a DateTime by parsing a string and based on parameters passed.
Currently, the following formats are supported:
(1) YYYYMMDD
AME_STYLE:
(2) DD/MM/YYYY
(3) DD/MM/YYYY HH
(4) DD/MM/YYYY HH:MM
(5) DD/MM/YYYY HH:MM:SS
(6) DD/MM/YYYY HH:MM:SS.MMM

EUR_STYLE:

(7) YYYY/MM/DD
(8) YYYY/MM/DD HH
(9) YYYY/MM/DD HH:MM
(10) YYYY/MM/DD HH:MM:SS
(11) YYYY/MM/DD HH:MM:SS.MMM

s: The string to be parsed
ds: String format from DT_DATE_STYLE above
the_zone: Desired time zone from DT_TIME_ZONE above.

void set_time (EpochType the_time, NanosecondType nanosec = 0) noexcept;
A convenient method, if you already have a DateTime instance and want to change the date/time quickly.

the_time: Time as epoch
nanosec: Nano seconds

void set_timezone (DT_TIME_ZONE tz);
Changes the time zone to desired time zone.
NOTE: This method is not multithread-safe. This method modifies the TZ environment variable which changes the time zone for the entire program.

tz: Desired time zone

DT_TIME_ZONE get_timezone () const;
Returns the current time zone.

DateTime &operator = (DateType rhs);
Sets self to right-hand-side.

rhs: A date e.g. dt = 20181215;

*DateTime &operator = (const char *rhs);*
Sets self to right-hand-side.
Currently, the following formats are supported:
1) YYYYMMDD [LOCAL | GMT]
2) YYYYMMDD HH:MM:SS.MMM [LOCAL | GMT]

rhs: A date/time string e.g. dt = “20181215”;

int dt_compare(const DateTime &rhs) const;
Compares self with right-hand-side and returns an integer result accordingly.

rhs: Another DateTime instance

```
DateType date () const noexcept;                    // e.g. 20020303
DatePartType year () const noexcept;                // e.g. 1990
DT_MONTH month () const noexcept;                   // JAN - DEC
DatePartType dmonth () const noexcept;              // 1 - 31
DatePartType dyear () const noexcept;               // 1 - 366
DT_WEEKDAY dweek () const noexcept;                 // SUN - SAT
HourType hour () const noexcept;                    // 0 - 23
MinuteType minute () const noexcept;                // 0 - 59
SecondType sec () const noexcept;                   // 0 - 59
MillisecondType msec () const noexcept;             // 0 - 999
MicrosecondType microsec () const noexcept;         // 0 - 999,999
NanosecondType nanosec () const noexcept;           // 0 - 999,999,999
EpochType time () const noexcept;                   // Like time()
```
These methods return the corresponding date/time parts.

```
DatePartType days_in_month () const noexcept;       // 28, 29, 30, 31
```
It returns the number of days in the month represented in self

```
double diff_seconds (const DateTime &that) const;
double diff_minutes (const DateTime &that) const noexcept;
double diff_hours (const DateTime &that) const noexcept;
double diff_days (const DateTime &that) const noexcept;
double diff_weekdays (const DateTime &that) const noexcept;
double diff_weeks (const DateTime &that) const noexcept;
```
These return the diff including the fraction of the unit. This is why they return a double. The diff could be +/- based on "this - that"

that: Another instance of DateTime

```
void add_seconds (EpochType secs) noexcept;
void add_days (long days) noexcept;
void add_weekdays (long days) noexcept;
```
These methods either advance or pullback the time accordingly. The parameter to these methods could be +/-.

secs, days: A positive or negative number representing the units to change time

```
template<typename T>
void date_to_str (DT_FORMAT format, T &result) const;
std::string string_format (DT_FORMAT format) const;
```
These methods format the date/time into a string based on the format parameter

T: Type of string
result: a string instance to store the formatted date/time
format: String format parameter based on DT_FORMAT above