Date and Time

Working with date and time, time spans, time zones as well as formatting and parsing date and time.

applied informatics

Overview

- Timestamp
- DateTime and LocalDateTime
- Timespan
- Timezone
- Formatting and parsing dates and times

The Timestamp Class

- Poco::Timestamp is the main time keeping class in Poco.
- #include "Poco/Timestamp.h"
- It stores a UTC-based ~~monotonic~~ 64-bit time value with (up to) microsecond resolution. The actual resolution depends on the operating system.
- Since Poco::Timestamp is UTC-based, it is independent of the timezone (and changes to it).
- Poco::Timestamp supports value semantics, comparison and simple arithmetics.

The Timestamp Class (cont'd)

- Poco::Timestamp defines a few public types:
  - TimeVal
    a 64-bit signed integer holding UTC time with microsecond resolution
  - UtcTimeVal
    a 64-bit signed integer holding UTC time with 100 nanoseconds resolution (actual resolution is still <= 1 µs)
  - TimeDiff
    a 64-bit signed integer holding the difference between two Timestamp's in microseconds

Epoch Time and UTC Time

- In Unix, epoch time is the time measured in seconds since midnight, January 1, 1970.
- UTC (Coordinated Universal Time) is the time measured in 100 nanosecond intervals since midnight, October 15, 1582.

Constructing a Timestamp

- The default constructor initializes a Timestamp with the current time.
- Two static functions can be used to create a Timestamp from a time_t, or from a UTC time:
  - Timestamp fromEpochTime(time_t time)
  - Timestamp fromUtcTime(UtcTimeVal val)

Timestamp Functions

- time_t epochTime() const
  returns the time expressed in time_t (epoch time)

- UtcTimeVal utcTime() const
  returns the time expressed in UTC with 100 nanoseconds resolution

- TimeVal epochMicroseconds() const
  returns the time expressed in microseconds since the Unix epoch

Timestamp Functions (cont'd)

- void update()
  updates the Timestamp with the current time
- TimeDiff elapsed() const
  returns the microseconds elapsed since the time stored in the Timestamp
- bool isElapsed(TimeDiff interval) const
  returns true if at least interval microseconds have passed since the time stored in the Timestamp

Timestamp Arithmetics

- Timestamp operator + (TimeDiff diff) const
  adds a time span to the Timestamp and returns the result

- Timestamp operator - (TimeDiff diff) const
  subtracts a time span from the Timestamp and returns the result

- TimeDiff operator - (const Timestamp& ts) const
  returns the time difference between two Timestamp's

- Timestamp& operator += (TimeDiff d)
  Timestamp& operator -= (TimeDiff d)
  adds/subtracts a time span to/from the Timestamp

```
#include "Poco/Timestamp.h"
#include <ctime>

using Poco::Timestamp;

int main(int argc, char** argv)
{
    Timestamp now; // the current date and time

    std::time_t t1 = now.epochTime();             // convert to time_t ...
    Timestamp ts1(Timestamp::fromEpochTime(t1)); // ... and back again

    for (int i = 0; i < 100000; ++i) ;            // wait a bit

    Timestamp::TimeDiff diff = now.elapsed();    // how long did it take?

    Timestamp start(now);                         // save start time
    now.update();                                 // update with current
    time

    diff = now - start;                           // again, how long?

    return 0;
}
```

The DateTime Class

- Poco::DateTime is used for working with calendar dates and times, based on the Gregorian calendar.
- #include "Poco/DateTime.h"
- It should be used for date calculations only. For storing dates and times, the Timestamp class is more appropriate.
- Internally, DateTime maintains the date and time in two formats: UTC and broken down (into year, month, day, hour, minute, second, millisecond, microsecond).
- For internal conversions, DateTime also uses the Julian Day.

The Gregorian Calendar

- The Gregorian calendar, a modification of the Julian calendar, is used nearly everywhere in the world. Its years are numbered based on the traditional birth year of Jesus Christ, which is labeled the "anno Domini" era.
- It counts days as the basic unit of time, grouping them into years of 365 or 366 days. A year is divided into 12 months of irregular length.
- Not all countries adopted the Gregorian calendar at the same time (e.g., Germany in 1582, England in 1752).
- Therefore, Gregorian dates and historic (old-style) dates might differ, although referring to the same event in time.

Julian Day and Julian Date

- The Julian day or Julian day number (JDN) is the number of days that have elapsed since Monday, January 1, 4713 BC in the proleptic Julian calendar. That day is counted as Julian day zero. Thus the multiples of 7 are Mondays. Negative values can also be used.

- The Julian Date (JD) is the number of days (with decimal fraction of the day) that have elapsed since 12 noon Greenwich Mean Time (UT or TT) of that day.

DateTime Considerations

- Zero is a valid year (in accordance with ISO 8601 and astronomical year numbering).
- Year zero is a leap year.
- Negative years (years preceding 1 BC) are not supported.
- Gregorian dates might differ from historical dates, depending on which calendar was in use at the time.
- It is best to use DateTime for "current" dates only. For historical or astronomical calendar calculations, specialized software (that takes into account local Gregorian calendar adoption, etc.) should be used.

Constructing a DateTime

- A DateTime can be constructed from:
  - the current date and time
  - a Timestamp
  - a broken-down date and time (year, month, day, hour, minute, second, millisecond, microsecond)
  - a Julian day (stored in a double)

DateTime Functions

- int year() const
  returns the year

- int month() const
  returns the month (1 - 12)

- int week(int firstDayOfWeek = DateTime::MONDAY) const
  returns the week number within the year, according to ISO 8601 (week 1 is the week containing January 4); firstDayOfWeek should be DateTime::MONDAY or DateTime::SUNDAY.

- int day() const
  returns the day within the month (1 - 31)

DateTime Functions (cont'd)

- int dayOfWeek() const
  returns the day within the week
  (0 = DateTime::SUNDAY, 1 = DateTime::MONDAY, ...,
  6 = DateTime::SATURDAY)

- int dayOfYear() const
  returns the number of the day in the year (1 - 366)

DateTime Functions (cont'd)

- int hour() const
  returns the hour (0 - 23)
- int hourAMPM() const
  returns the hour (0 - 12)
- bool isAM() const
  returns true if hour() < 12, false otherwise
- bool isPM() const
  returns true if hour() >= 12, false otherwise

DateTime Functions (cont'd)

- int minute() const
  returns the minute (0 - 59)
- int second() const
  returns the second (0 - 59)
- int millisecond() const
  returns the millisecond (0 - 999)
- int microsecond() const
  returns the microsecond (0 - 999)

DateTime Functions (cont'd)

- Timestamp timestamp() const
  returns the date and time expressed as a Timestamp
- Timestamp::UtcTimeVal utcTime() const
  returns the date and time expressed in UTC-based time
- DateTime supports all relational operators
  (==, !=, >, >=, <, <=).
- DateTime supports arithmetics like Timestamp (+, -, +=, -=)

DateTime Static Functions

- bool isLeapYear(int year)
  returns true if the given year is a leap year, false otherwise

- int daysOfMonth(int year, int month)
  returns the number of days in the given month, for the given year

- bool isValid(int year, int month, int day,
  int hour, int minute, int second, int millisecond, int microsecond)
  returns true if the given date and time is valid (all arguments are within their proper ranges), false otherwise (takes into account leap years)

DateTime Month and Weekday Names

- Poco::DateTime has enumerations for month and weekday names. These can be used in place of numeric values:
  - DateTime::Months
    (JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER)
  - DateTime::DaysOfWeek
    (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY)

```
#include "Poco/DateTime.h"

using Poco::DateTime;

int main(int argc, char** argv)
{
    DateTime now; // the current date and time in UTC
    int year   = now.year();
    int month  = now.month();
    int day    = now.day();
    int dow    = now.dayOfWeek();
    int doy    = now.dayOfYear();
    int hour   = now.hour();
    int hour12 = now.hourAMPM();
    int min    = now.minute();
    int sec    = now.second();
    int ms     = now.millisecond();
    int us     = now.microsecond();
    double jd  = now.julianDay();
    Poco::Timestamp ts = now.timestamp();

    DateTime xmas(2006, 12, 25);                    // 2006-12-25 00:00:00
    Poco::Timespan timeToXmas = xmas - now;
```

```
DateTime dt(1973, 9, 12, 2, 30, 45);       // 1973-09-12 02:30:45
dt.assign(2006, 10, 13, 13, 45, 12, 345); // 2006-10-13 12:45:12.345

bool isAM = dt.isAM(); // false
bool isPM = dt.isPM(); // true

bool isLeap = DateTime::isLeapYear(2006);      // false
int days    = DateTime::daysOfMonth(2006, 2); // 28

bool isValid = DateTime::isValid(2006, 02, 29); // false

dt.assign(2006, DateTime::OCTOBER, 22); // 2006-10-22 00:00:00
    if (dt.dayOfWeek() == DateTime::SUNDAY)
    {
        // ...
    }

    return 0;
}
```

The LocalDateTime Class

- Poco::LocalDateTime is similar to Poco::DateTime, except that it stores a local time, as opposed to UTC, as well as a time zone differential.
- #include "Poco/LocalDateTime.h"
- The time zone differential denotes the difference in seconds between UTC and local time
(UTC = local time - time zone differential).

Constructing a LocalDateTime

- A LocalDateTime can be constructed from:
  - the current date and time
  - a Timestamp
  - a broken-down date and time (year, month, day, hour, minute, second, millisecond, microsecond)
  - a Julian day (stored in a double)
- Optionally, a time zone differential can be specified as first argument (if none is specified, the system's current time zone is used).

LocalDateTime Functions

- LocalDateTime supports all functions that DateTime supports.
- All relational operators normalize to UTC before carrying out the comparison.
- int tzd() const
  returns the time zone differential (seconds)
- DateTime utc() const
  converts the local time to UTC

```
#include "Poco/LocalDateTime.h"

using Poco::LocalDateTime;

int main(int argc, char** argv)
{
    LocalDateTime now; // the current date and local time
    int year   = now.year();
    int month  = now.month();
    int day    = now.day();
    int dow    = now.dayOfWeek();
    int doy    = now.dayOfYear();
    int hour   = now.hour();
    int hour12 = now.hourAMPM();
    int min    = now.minute();
    int sec    = now.second();
    int ms     = now.millisecond();
    int us     = now.microsecond();
    int tzd    = now.tzd();
    double jd  = now.julianDay();
    Poco::Timestamp ts = now.timestamp();
```

```
	LocalDateTime dt1(1973, 9, 12, 2, 30, 45); // 1973-09-12 02:30:45
	dt1.assign(2006, 10, 13, 13, 45, 12, 345); // 2006-10-13 12:45:12.345

	LocalDateTime dt2(3600, 1973, 9, 12, 2, 30, 45, 0, 0); // UTC +1 hour
	dt2.assign(3600, 2006, 10, 13, 13, 45, 12, 345, 0);

	Poco::Timestamp nowTS;
	LocalDateTime dt3(3600, nowTS); // construct from Timestamp

	return 0;
}
```

The Timespan Class

- Poco::Timespan represents a time span up to microsecond resolution.
- #include "Poco/Timespan.h"
- Internally, Poco::Timespan uses a 64-bit integer to store the time span.
- The time span can be expressed in days, hours, minutes, seconds, milliseconds and microseconds.

Timespan Scaling Factors

- Poco::Timespan defines the following scaling factors:
  - MILLISECONDS
    the number of microseconds in a millisecond
  - SECONDS
    the number of microseconds in a second
  - MINUTES
    the number of microseconds in a minute
  - HOURS
    the number of microseconds in a hour
  - DAYS
    the number of microseconds in a day

Constructing a Timespan

- A Timespan can be constructed from:
  - a TimeStamp::TimeDiff (microseconds)
  - seconds + microseconds
    useful for constructing a Timespan from a struct timeval
  - broken down days, hours, minutes, seconds, microseconds

Timespan Operators

- Poco::Timespan supports all relational operators (==, !=, <, <=, >, >=)
- Poco::Timespan supports addition and subtraction (+, -, +=, -=)

Timespan Functions

- int days() const
  returns the number of days
- int hours() const
  return the number of hours within the day (0 - 23)
- int totalHours() const
  returns the total number of hours
- int minutes() const
  return the number of minutes within the hour (0 - 59)
- int totalMinutes() const
  return the total number of minutes

Timespan Functions (cont'd)

- int seconds() const
  returns the number of seconds within the minute (0 - 60)
- int totalSeconds() const
  returns the total number of seconds
- int milliseconds() const
  returns the number of milliseconds within the second (0 - 999)
- int totalMilliseconds() const
  returns the total number of milliseconds

Timespan Functions (cont'd)

- int microseconds() const
  returns the number of microseconds within the millisecond (0 - 999)
- int totalMicroseconds() const
  returns the total number of microseconds

```
#include "Poco/Timespan.h"

using Poco::Timespan;

int main(int argc, char** argv)
{
    Timespan ts1(1, 11, 45, 22, 123433); // 1d 11h 45m 22.123433s
    Timespan ts2(33*Timespan::SECONDS);  // 33s
    Timespan ts3(2*Timespan::DAYS + 33*Timespan::HOURS); // 3d 33h

    int days       = ts1.days();          // 1
    int hours      = ts1.hours();         // 11
    int totalHours = ts1.totalHours();    // 35
    int minutes    = ts1.minutes();       // 45
    int totalMins  = ts1.totalMinutes(); // 2145
    int seconds    = ts1.seconds();       // 22
    int totalSecs  = ts1.totalSeconds(); // 128722

    return 0;
}
```

DateTime and Timespan Calculations

- DateTime, LocalDateTime and Timespan can be used together for date and time arithmetics, e.g.:
  - add a number of days to a date
  - add a number of hours to a date
  - calculate the difference between two dates in days (or hours, or seconds, ...)

```
#include "Poco/DateTime.h"
#include "Poco/Timespan.h"

using Poco::DateTime;
using Poco::Timespan;

int main(int argc, char** argv)
{
    // what is my age?
    DateTime birthdate(1973, 9, 12, 2, 30); // 1973-09-12 02:30:00
    DateTime now;
    Timespan age = now - birthdate;
    int days  = age.days();          // in days
    int hours = age.totalHours();    // in hours
    int secs  = age.totalSeconds(); // in seconds

    // when was I 10000 days old?
    Timespan span(10000*Timespan::DAYS);
    DateTime dt = birthdate + span;

    return 0;
}
```

The Timezone Class

- Poco::Timezone provides static methods for getting information about the system's time zone, such as:
  - the time zone differential
  - whether daylight saving time (DST) is in effect
  - the name of the time zone
- #include "Poco/Timezone.h"

Timezone Functions

- int utcOffset()
  returns the offset of local time to UTC, in seconds, and not including DST (local time = UTC + utcOffset())

- int dst()
  returns the daylight saving time offset in seconds (usually 3600) if DST is in effect, or 0 otherwise.

- bool isDst(const Timestamp& timestamp)
  returns true if DST is in effect for the given time

- int tzd()
  returns the time zone differential for the current time zone (tzd = utcOffset() + dst())

Timezone Functions (cont'd)

- std::string name()
  returns the time zone's name currently in effect
- std::string standardName()
  returns the time zone's name if DST is not in effect
- std::string dstName()
  returns the time zone's name if DST is in effect
- The names reported are dependent on the operating system and are not portable across systems.
  They should be used for display purposes only.

```
#include "Poco/Timezone.h"
#include "Poco/Timestamp.h"

using Poco::Timezone;
using Poco::Timestamp;

int main(int argc, char** argv)
{
    int utcOffset = Timezone::utcOffset();
    int dst       = Timezone::dst();
    bool isDst    = Timezone::isDst(Timestamp());
    int tzd       = Timezone::tzd();

    std::string name    = Timezone::name();
    std::string stdName = Timezone::standardName();
    std::string dstName = Timezone::dstName();

    return 0;
}
```

Formatting Date and Time

- Poco::DateTimeFormatter can be used to format dates and times (Timestamp, DateTime, LocalDateTime and Timespan) as strings.
- #include "Poco/DateTimeFormat.h"
- Poco::DateTimeFormatter uses a format string similar to strftime(). For details, please see the reference documentation.
- Pre-defined format strings for commonly used formats can be found in class Poco::DateTimeFormat.

DateTimeFormatter Functions

- All functions of DateTimeFormatter are static.
- std::string format(const Timestamp& timestamp, const std::string& fmt, int tzd = UTC)
  formats the given Timestamp according to the given format string fmt. The time zone differential (tzd) is optional.
- std::string format(const DateTime& dateTime, const std::string& fmt, int tzd = UTC)
  is similar to the previous function, except that this one accepts a DateTime

DateTimeFormatter Functions (cont'd)

- std::string format(const LocalDateTime& dateTime,
  const std::string& fmt)
  accepts a LocalDateTime (which includes a time zone differential)
  and formats it according to the format string fmt

- std::string format(const Timespan& timespan,
  const std::string& fmt)
  formats the given timespan according to the format string fmt

DateTimeFormatter Functions (cont'd)

- All format() functions have append() counterparts that should be used to improve performance when appending a formatted date/time to a string.
- All format() functions are implemented using append().

Pre-defined Format Strings

- Poco::DateTimeFormat defines a set of commonly used date and time formats.
- #include "Poco/DateTimeFormat.h"
- Examples:
  - ISO8601_FORMAT (2005-01-01T12:00:00+01:00)
  - RFC1123_FORMAT (Sat, 1 Jan 2005 12:00:00 +0100)
  - SORTABLE_FORMAT (2005-01-01 12:00:00)
- For more information, please see the reference documentation.

```
#include "Poco/DateTime.h"
#include "Poco/Timestamp.h"
#include "Poco/Timespan.h"
#include "Poco/DateTimeFormatter.h"
#include "Poco/DateTimeFormat.h"

using Poco::DateTimeFormatter;
using Poco::DateTimeFormat;

int main(int argc, char** argv)
{
    Poco::DateTime dt(2006, 10, 22, 15, 22, 34);
    std::string s(DateTimeFormatter::format(dt, "%e %b %Y %H:%M"));
        // "22 Oct 2006 15:22"

    Poco::Timestamp now;
    s = DateTimeFormatter::format(now, DateTimeFormat::SORTABLE_FORMAT);
        // "2006-10-30 09:27:44"

    Poco::Timespan span(5, 11, 33, 0, 0);
    s = DateTimeFormatter::format(span, "%d days, %H hours, %M minutes");
        // "5 days, 11 hours, 33 minutes"

    return 0;
}
```

Parsing Date and Time

- Poco::DateTimeParser can be used to parse dates and times from strings.
- #include "Poco/DateTimeParser.h"
- Poco::DateTimeParser always returns a DateTime and a time zone differential. The DateTime can then be converted to a Timestamp or a LocalDateTime.
- All functions of Poco::DateTimeParser are static.
- Poco::DateTimeParser uses the same format strings as Poco::DateTimeFormatter.

DateTimeParser Functions

- void parse(const std::string fmt, const std::string& str, DateTime& dateTime, int& tzd)

  parses a date and time in the format given in fmt from the string str. Stores the date and the time in dateTime and the time zone differential in tzd. Throws a Poco::SyntaxException if the string cannot be parsed.

- DateTime parse(const std::string& fmt, const std::string& str, int& tzd)

  similar to the above, but returns the DateTime

DateTimeParser Functions (cont'd)

- bool tryParse(const std::string& fmt, const std::string& str, DateTime& dateTime, int& tzd)
  tries to parse a date and time in the format given in fmt from the string str. If successful, stores the date and the time in dateTime and the time zone differential in tzd. Returns true if successful, otherwise false.

DateTimeParser Functions (cont'd)

- void parse(const std::string& str, DateTime& dateTime, int& tzd)
  parses a date and a time from the given string, recognizing all standard date/time formats defined by Poco::DateTimeFormat. Throws a Poco::SyntaxException if the format cannot be recognized, or the string does not match the expected format.
- DateTime parse(const std::string& str, int& tzd)
- bool tryParse(const std::string& str, DateTime& dateTime, int& tzd)

```
#include "Poco/DateTimeParser.h"
#include "Poco/DateTime.h"
#include "Poco/DateTimeFormat.h"
#include "Poco/LocalDateTime.h"
#include "Poco/Timestamp.h"

using Poco::DateTimeParser;
using Poco::DateTimeFormat;
using Poco::DateTime;

int main(int argc, char** argv)
{
	std::string s("Sat, 1 Jan 2005 12:00:00 GMT");
	int tzd;
	DateTime dt;
	DateTimeParser::parse(DateTimeFormat::RFC1123_FORMAT, s, dt, tzd);
	Poco::Timestamp ts = dt.timestamp();
	Poco::LocalDateTime ldt(tzd, dt);

	bool ok = DateTimeParser::tryParse("2006-10-22", dt, tzd);
	ok = DateTimeParser::tryParse("%e.%n.%Y", "22.10.2006", dt, tzd);

	return 0;
}
```

To dig deeper...

- The Gregorian Calendar
  http://en.wikipedia.org/wiki/Gregorian_calendar
- The Julian Calendar
  http://en.wikipedia.org/wiki/Julian_calendar
- The Julian Day
  http://en.wikipedia.org/wiki/Julian_day
- Coordinated Universal Time (UTC)
  http://en.wikipedia.org/wiki/UTC
- ISO 8601
  http://en.wikipedia.org/wiki/ISO_8601

applied informatics

www.appinf.com | info@appinf.com
T +43 4253 32596 | F +43 4253 32096