HIC for FAIR
Helmholtz International Center

std::chrono – typesafe time keeping in C++

Bjørn Bäuchle

FIAS

May 28, 2014

Outline

C variant

The C++ ansatz: std::chrono

- Clocks
- Durations
- Time Points
- Dates

Recapitulation

<time.h>

Clock: count ticks

```
clock_t a, b;
a = clock();
do_something_long(with, variables);
b = clock() - a;
printf("took␣me␣%d␣clicks␣=␣%f␣seconds", b,
        ((float b)/CLOCKS_PER_SECOND);
```

Output

took me 143 clicks = 0.143 seconds

<time.h>

Clock: count ticks

```
clock_t a, b;
a = clock();
do_something_long(with, variables);
b = clock() - a;
printf("took␣me␣%d␣clicks␣=␣%f␣seconds", b,
        ((float b)/CLOCKS_PER_SECOND);
```

Output

took me 143 clicks = 0.143 seconds

<time.h>

Clock: count ticks

```
clock_t a, b;
a = clock();
do_something_long(with, variables);
b = clock() - a;
printf("took␣me␣%d␣clicks␣=␣%f␣seconds", b,
        ((float b)/CLOCKS_PER_SECOND);
```

Output

took me 143 clicks = 0.143 seconds

<time.h>

Clock: count ticks

```
clock_t a, b;
a = clock();
do_something_long(with, variables);
b = clock() - a;
printf("took␣me␣%d␣clicks␣=␣%f␣seconds", b,
       ((float b)/CLOCKS_PER_SECOND);
```

Output

took me 143 clicks = 0.143 seconds

<time.h> cont'd.

Time: returns seconds in epoch (since Jan 1 1970)

```
time_t now, also_now;                tm * local_time;
now = time(&also_now);
local_time = localtime(&now);
printf("Now: %s", asctime(local_time));
```

Output

```
Now: Wed May 28 15:05:45 2014
```

tm is struct of 9 ints with weekday, month, minutes etc.

<time.h> cont'd.

Time: returns seconds in epoch (since Jan 1 1970)

```
time_t now, also_now;                    tm * local_time;
now = time(&also_now);
local_time = localtime(&now);
printf("Now: %s", asctime(local_time));
```

Output

Now: Wed May 28 15:05:45 2014

tm is struct of 9 ints with weekday, month, minutes etc.

<time.h> cont'd.

Time: returns seconds in epoch (since Jan 1 1970)

```
time_t now, also_now;                    tm * local_time;
now = time(&also_now);
local_time = localtime(&now);
printf("Now: %s", asctime(local_time));
```

Output

```
Now: Wed May 28 15:05:45 2014
```

tm is struct of 9 ints with weekday, month, minutes etc.

<time.h> cont'd.

Time: returns seconds in epoch (since Jan 1 1970)

```
time_t now, also_now;                    tm * local_time;
now = time(&also_now);
local_time = localtime(&now);
printf("Now: %s", asctime(local_time));
```

Output

```
Now: Wed May 28 15:05:45 2014
```

tm is struct of 9 ints with weekday, month, minutes etc.

<time.h> cont'd.

timespec: High precision time measurements

```
timespec a, b;
clock_gettime(CLOCK_REALTIME, &a);
do_something_long(with, variables);
clock_gettime(CLOCK_REALTIME, &b);
print("Took␣me␣%d␣seconds␣and␣%d␣nanoseconds",
    b.tv_sec - a.tv_sec, b.tv_nsec - a.tv_nsec);
```

Output

Took me 15 seconds and 158720 nanoseconds

<time.h> cont'd.

timespec: High precision time measurements

```
timespec a, b;
clock_gettime(CLOCK_REALTIME, &a);
do_something_long(with, variables);
clock_gettime(CLOCK_REALTIME, &b);
print("Took␣me␣%d␣seconds␣and␣%d␣nanoseconds",
    b.tv_sec - a.tv_sec, b.tv_nsec - a.tv_nsec);
```

Output

Took me 15 seconds and 158720 nanoseconds

CLOCK_REALTIME guaranteed to exist; other clocks: interpretation non-defined.

<time.h> cont'd.

timespec: High precision time measurements

```
timespec a, b;
clock_gettime(CLOCK_REALTIME, &a);
do_something_long(with, variables);
clock_gettime(CLOCK_REALTIME, &b);
print("Took␣me␣%d␣seconds␣and␣%d␣nanoseconds",
    b.tv_sec - a.tv_sec, b.tv_nsec - a.tv_nsec);
```

Output

Took me 15 seconds and 158720 nanoseconds

CLOCK_REALTIME guaranteed to exist; other clocks: interpretation non-defined.

C-Style: No operator overloading for timespec - timespec

What we try to avoid

In C-header <time.h>:

```
struct timespec
  {
    __time_t tv_sec;          /* Seconds.  */
    long int tv_nsec;         /* Nanoseconds.  */
  };
```

Real-life C code (experienced kernel programmer)

```
/* timediff - time the simulation used */
double timediff(const timespec time_start) {
 timespec now;
 clock_gettime(CLOCK_REALTIME, &now);
 return (now.tv_sec+now.tv_nsec/10.0E9
 - time_start.tv_sec-time_start.tv_nsec/10.0E9);
}
```

What we try to avoid

In C-header <time.h>:

```
struct timespec
  {
    __time_t tv_sec;          /* Seconds.   */
    long int tv_nsec;         /* Nanoseconds.   */
  };
```

Real-life C code (experienced kernel programmer)

```
/* timediff - time the simulation used */
double timediff(const timespec time_start) {
 timespec now;
 clock_gettime(CLOCK_REALTIME, &now);
 return (now.tv_sec+now.tv_nsec/10.0E9
 - time_start.tv_sec-time_start.tv_nsec/10.0E9);
}
```

What we try to avoid

In C-header <time.h>:

```
struct timespec
  {
    __time_t tv_sec;            /* Seconds.   */
    long int tv_nsec;           /* Nanoseconds.   */
  };
```

Real-life C code (experienced kernel programmer)

```
/* timediff - time the simulation used */
double timediff(const timespec time_start) {
 timespec now;
 clock_gettime(CLOCK_REALTIME, &now);
 return (now.tv_sec+now.tv_nsec/10.0E9
 - time_start.tv_sec-time_start.tv_nsec/10.0E9);
}
```

Problem: $10.0E9 = 10^{10} \neq 10^9$

What we try to avoid

In C-header <time.h>:

```
struct timespec
  {
    __time_t tv_sec;            /* Seconds.   */
    long int tv_nsec;           /* Nanoseconds.   */
  };
```

Real-life C code (experienced kernel programmer)

```
/* timediff - time the simulation used */
double timediff(const timespec time_start) {
 timespec now;
 clock_gettime(CLOCK_REALTIME, &now);
 return (now.tv_sec+now.tv_nsec/10.0E9
 - time_start.tv_sec-time_start.tv_nsec/10.0E9);
}
```

Problem: $10.0E9 = 10^{10} \neq 10^9$

Task: make this type-safe!

C variant

The C++ ansatz: std::chrono

- Clocks
- Durations
- Time Points
- Dates

Recapitulation

std::chrono::*: A comprehensive library

Three fundamental concepts

- Durations: Count representation + period precision
- Time Points: equiv. to duration relative to epoch (fixed time point that is a property of current clock)
- Clocks: relate Time Points to real physical time.

std::chrono::*: A comprehensive library

Three fundamental concepts

- Durations: Count representation + period precision
- Time Points: equiv. to duration relative to epoch (fixed time point that is a property of current clock)
- Clocks: relate Time Points to real physical time.

std::chrono::*: A comprehensive library

Three fundamental concepts

- Durations: Count representation + period precision
- Time Points: equiv. to duration relative to epoch (fixed time point that is a property of current clock)
- Clocks: relate Time Points to real physical time.

Clocks

Base unit always one second – not well applicable for e.g. heavy ion collisions

system_clock

- meant to represent real time
- system-wide

Clocks

Base unit always one second – not well applicable for e.g. heavy ion collisions

system_clock

- meant to represent real time
- system-wide

steady_clock

- guaranteed to be monotonic
- steady: each tick has same duration

Clocks

Base unit always one second – not well applicable for e.g. heavy ion collisions

system_clock

- meant to represent real time
- system-wide

steady_clock

- guaranteed to be monotonic
- steady: each tick has same duration

high_resolution_clock

- guaranteed to have highest available precision

Durations: Ratios

Durations are represented by count and period; period is a ratio.

Ratios: compile-time numerator / denominator

```
typedef std::ratio<2, 4> t_4;
std::cout << "2/4␣␣␣=␣" << t_4::num << "/"
                        << t_4::den << std::endl;
    << "kilo␣␣=␣" << std::kilo::num << "/"
                  << std::kilo::den << std::endl;
   << "centi␣=␣" << std::centi::num << "/"
                 << std::centi::den << std::endl;
```

Output:

```
2/4   = 1/2
kilo  = 1000/1
centi = 1/100
```

SI decimal prefixes pre-defined (deca, hecto, kilo, mega, giga, tera etc.; also deci, centi, milli, nano, micro, nano, pico etc.)

Durations: Special ratios

Time ratios

```
using namespace std;
std::chrono::duration<int, std::chrono::hours>
                                oneday_in_hrs(24);
std::chrono::duration<int, std::chrono::milliseconds>
                    oneday_in_ms(oneday_in_hrs);
std::cout << oneday_in_hrs.count() << " hours/day = "
     << oneday_in_ms.count()  << " milliseconds";
```

Output:

```
24 hours/day = 86400000 milliseconds
```

<chrono> defines (in std::chrono) hours, minutes, seconds, milliseconds, microseconds and nanoseconds.

Durations: Special ratios

Time ratios

```
using namespace std;
std::chrono::duration<int, std::chrono::hours>
                                oneday_in_hrs(24);
std::chrono::duration<int, std::chrono::milliseconds>
                    oneday_in_ms(oneday_in_hrs);
std::cout << oneday_in_hrs.count() << " hours/day = "
     << oneday_in_ms.count()  << " milliseconds";
```

Output:

```
24 hours/day = 86400000 milliseconds
```

<chrono> defines (in std::chrono) hours, minutes, seconds, milliseconds, microseconds and nanoseconds.
chrono::duration's first template parameter (here: int) defines internal tick counter

Durations: Arithmetic

Multiply, add, divide

```
std::chrono::minutes m(31);
m *= 2;                                  // m =  1:02 hrs
m += std::chrono::hours(10);             // m = 11:02 hrs
std::cout << m.count() << " minutes equals "
  << std::chrono::duration_cast<
          std::chrono::hours>(m).count()
  << " hours and ";
m %= std::chrono::hours(1);
std::cout << m.count() << " minutes";
```

Output

```
662 minutes equals 11 hours and 2 minutes
```

Durations: Arithmetic

Multiply, add, divide

```
std::chrono::minutes m(31);
m *= 2;                                  // m =  1:02 hrs
m += std::chrono::hours(10);             // m = 11:02 hrs
std::cout << m.count() << " minutes equals "
  << std::chrono::duration_cast<
          std::chrono::hours>(m).count()
  << " hours and ";
m %= std::chrono::hours(1);
std::cout << m.count() << " minutes";
```

Output

```
662 minutes equals 11 hours and 2 minutes
```

type-safe addition of durations with different units!

Durations: Arithmetic

Multiply, add, divide

```
std::chrono::minutes m(31);
m *= 2;                                  // m =  1:02 hrs
m += std::chrono::hours(10);             // m = 11:02 hrs
std::cout << m.count() << " minutes equals "
  << std::chrono::duration_cast<
          std::chrono::hours>(m).count()
  << " hours and ";
m %= std::chrono::hours(1);
std::cout << m.count() << " minutes";
```

Output

```
662 minutes equals 11 hours and 2 minutes
```

type-safe addition of durations with different units!
duration_cast changes the base of the duration

Durations: Arithmetic

Multiply, add, divide

```
std::chrono::minutes m(31);
m *= 2;                                  // m =  1:02 hrs
m += std::chrono::hours(10);             // m = 11:02 hrs
std::cout << m.count() << " minutes equals "
  << std::chrono::duration_cast<
          std::chrono::hours>(m).count()
  << " hours and ";
m %= std::chrono::hours(1);
std::cout << m.count() << " minutes";
```

Output

```
662 minutes equals 11 hours and 2 minutes
```

type-safe operations of durations with different units!
duration_cast changes the base of the duration

Durations: Casts

duration_cast

```
typedef std::chrono::duration<float, std::ratio<
          24*60*60*14, 1000000>> microfortnights;
// lott = length of this talk
std::chrono::minutes lott(30);
// in C++14, you can also do
std::chrono::minutes lott = 30_min;
// or
std::chrono::minutes lott = 1800_sec;
std::cout << "This␣talk␣takes␣approx␣"
          << microfortnights(lott).count()
          << "␣microfortnights";
```

Output

```
This talk takes approx 1488.09523 microfortnights
```

Durations: Casts

duration_cast

```
typedef std::chrono::duration<float, std::ratio<
          24*60*60*14, 1000000>> microfortnights;
// lott = length of this talk
std::chrono::minutes lott(30);
// in C++14, you can also do
std::chrono::minutes lott = 30_min;
// or
std::chrono::minutes lott = 1800_sec;
std::cout << "This␣talk␣takes␣approx␣"
          << microfortnights(lott).count()
          << "␣microfortnights";
```

Output

```
This talk takes approx 1488.09523 microfortnights
```

Internal counter need not be integral type!

Durations: Casts

duration_cast

```
typedef std::chrono::duration<float, std::ratio<
          24*60*60*14, 1000000>> microfortnights;
// lott = length of this talk
std::chrono::minutes lott(30);
// in C++14, you can also do
std::chrono::minutes lott = 30_min;
// or
std::chrono::minutes lott = 1800_sec;
std::cout << "This talk takes approx "
          << microfortnights(lott).count()
          << " microfortnights";
```

Output

```
This talk takes approx 1488.09523 microfortnights
```

Internal counter need not be integral type!

Base unit still one second

Time Points

Construct, Assign and Cast

```
using SC = std::chrono;
SC::time_point<SC::high_resolution_clock,
               SC::seconds> Sec(5);
SC::time_point<SC::high_resolution_clock,
               SC::milliseconds> mSec(Sec);
mSec = SC::time_point<SC::high_resolution_clock,
                      SC::milliseconds>(2852014);
// Sec = mSec;     // will fail!
Sec = SC::time_point_cast<SC::high_resolution_clock,
                          SC::seconds>(mSec);
```

Time Points

Construct, Assign and Cast

```
using SC = std::chrono;
SC::time_point<SC::high_resolution_clock,
               SC::seconds> Sec(5);
SC::time_point<SC::high_resolution_clock,
               SC::milliseconds> mSec(Sec);
mSec = SC::time_point<SC::high_resolution_clock,
                      SC::milliseconds>(2852014);
// Sec = mSec;     // will fail!
Sec = SC::time_point_cast<SC::high_resolution_clock,
                          SC::seconds>(mSec);
```

Cast seconds $\Rightarrow$ milliseconds is always possible

Time Points

Construct, Assign and Cast

```
using SC = std::chrono;
SC::time_point<SC::high_resolution_clock,
               SC::seconds> Sec(5);
SC::time_point<SC::high_resolution_clock,
               SC::milliseconds> mSec(Sec);
mSec = SC::time_point<SC::high_resolution_clock,
                      SC::milliseconds>(2852014);
// Sec = mSec;     // will fail!
Sec = SC::time_point_cast<SC::high_resolution_clock,
                          SC::seconds>(mSec);
```

Cast seconds ⇒ milliseconds is always possible

Case milliseconds ⇒ seconds may lose precision: Must be explicit!

(These cast rules also apply to duration_casts)

Date handling?

Only via <ctime>

```
std::chrono::system_clock::time_point tp =
                    std::chrono::system_clock::now();
std::time_t =
          std::chrono::system_clock::to_time_t(tp);
std::cout << std::put_time(std::localtime(&tp),
                                                "%F␣%T");
```

Output

```
2014-28-05 15:21:45
```

Date handling?

Only via <ctime>

```
std::chrono::system_clock::time_point tp =
                    std::chrono::system_clock::now();
std::time_t =
          std::chrono::system_clock::to_time_t(tp);
std::cout << std::put_time(std::localtime(&tp),
                                              "%F␣%T");
```

Output

```
2014-28-05 15:21:45
```

to_time_t converts time points to C-like std::time_t

Date handling?

Only via <ctime>

```
std::chrono::system_clock::time_point tp =
                    std::chrono::system_clock::now();
std::time_t =
          std::chrono::system_clock::to_time_t(tp);
std::cout << std::put_time(std::localtime(&tp),
                                          "%F␣%T");
```

Output

```
2014-28-05 15:21:45
```

to_time_t converts time points to C-like std::time_t
localtime converts time_t to std::tm

Date handling?

Only via <ctime>

```
std::chrono::system_clock::time_point tp =
                    std::chrono::system_clock::now();
std::time_t =
          std::chrono::system_clock::to_time_t(tp);
std::cout << std::put_time(std::localtime(&tp),
                                                "%F␣%T");
```

Output

```
2014-28-05 15:21:45
```

to_time_t converts time points to C-like std::time_t
localtime converts time_t to std::tm
put_time is like a printf for times

Reading dates (also <ctime>)

get_time

```
std::tm t;
std::istringstream ss("2014-Mai-28␣15:24:08");
ss.imbue(std::locale("de_DE"));
ss >> std::get_time(&t, "%Y-%b-%d␣%H:%M:%S");
std::cout << std::put_time(&t, "%T␣on␣%F");
```

Output

```
15:24:08 on 2014-28-05
```

Reading dates (also <ctime>)

get_time

```
std::tm t;
std::istringstream ss("2014-Mai-28␣15:24:08");
ss.imbue(std::locale("de_DE"));
ss >> std::get_time(&t, "%Y-%b-%d␣%H:%M:%S");
std::cout << std::put_time(&t, "%T␣on␣%F");
```

Output

```
15:24:08 on 2014-28-05
```

set-up: Stringstream and setting the locale

Reading dates (also <ctime>)

get_time

```
std::tm t;
std::istringstream ss("2014-Mai-28␣15:24:08");
ss.imbue(std::locale("de_DE"));
ss >> std::get_time(&t, "%Y-%b-%d␣%H:%M:%S");
std::cout << std::put_time(&t, "%T␣on␣%F");
```

Output

```
15:24:08 on 2014-28-05
```

set-up: Stringstream and setting the locale

get_time converts stringstream to std::tm object

Reading dates (also <ctime>)

get_time

```
std::tm t;
std::istringstream ss("2014-Mai-28␣15:24:08");
ss.imbue(std::locale("de_DE"));
ss >> std::get_time(&t, "%Y-%b-%d␣%H:%M:%S");
std::cout << std::put_time(&t, "%T␣on␣%F");
```

Output

```
15:24:08 on 2014-28-05
```

set-up: Stringstream and setting the locale
get_time converts stringstream to std::tm object
put_time for output

Converting time_t and tm to time_points

Connecting both worlds

```
std::tm timeinfo = std::tm();
// fill timeinfo...
std::time_t tt = std::mktime(&timeinfo);
std::chrono::system_clock::time_point tp =
      std::chrono::system_clock::from_time_t(tt);
// do calculations with time_point
```

Converting time_t and tm to time_points

Connecting both worlds

```
std::tm timeinfo = std::tm();
// fill timeinfo...
std::time_t tt = std::mktime(&timeinfo);
std::chrono::system_clock::time_point tp =
      std::chrono::system_clock::from_time_t(tt);
// do calculations with time_point
```

C style time things (imported to std::)

Converting time_t and tm to time_points

Connecting both worlds

```
std::tm timeinfo = std::tm();
// fill timeinfo...
std::time_t tt = std::mktime(&timeinfo);
std::chrono::system_clock::time_point tp =
      std::chrono::system_clock::from_time_t(tt);
// do calculations with time_point
```

C style time things (imported to std::)
from_time_t converts time_t to time_point (remember to_time_t)

C variant

The C++ ansatz: std::chrono

- Clocks
- Durations
- Time Points
- Dates

Recapitulation

Recapitulation

<chrono>

- Durations
- Time Points
- Clocks

<ctime>

- tm, time_t
- put_time, get_time (these are from <iomanip>, actually)
- mk_time, localtime, gmttime, asctime

Connection

- to_time_t, from_time_t

(and, btw, `ratios`.)