Google

Time Programming Fundamentals

Greg Miller (jgm@google.com)

Slides: goo.gl/ofof4N

Outline

- Vocabulary
- Classic time programming
- A simplified mental model
- Final thoughts
- Q&A

Vocabulary

Vocabulary

- Civil Time
- UTC
- Absolute Time
- Time Zone

Civil Time

- 6 fields: Year, Month, Day, Hour, Minute, Second
- "Normal human time"
- Gregorian calendar
- Examples:
  - std::tm
  - 6 separate ints
  - "2015-01-02 03:04:05"

YYYY-MM-DD

HH:MM:SS

UTC — International Time Standard

- Basis for local civil times worldwide
- No Daylight-Saving Time (DST)
- Uses the Gregorian calendar
- Ticks SI seconds
- Uses leap seconds

Source	Initials	Words
English	CUT	Coordinated Universal Time
French	TUC	Temps Universel Coordonné
Compromise	**UTC**	Unofficial English: "Universal Time Coordinated"; Unofficial French: "Universel Temps Coordonné"

https://en.wikipedia.org/wiki/Coordinated_Universal_Time#Etymology

Absolute Time

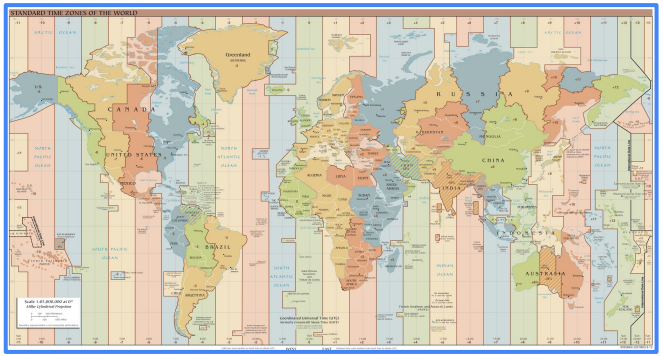
- Uniquely and universally represents a specific instant in time
- Time zone independent
- Count of `$unit` since `$epoch`
- Examples:
  - std::time_t
  - std::chrono::system_clock::time_point
  - int

Absolute Time

Civil Time

Time Zone

- Rules for converting between absolute times and civil times
- Rules defined using offsets from UTC
- Rules may change over time
- Rules established by local governments
- Rules may use Daylight-Saving Time (DST)



~~"PST"~~

"Europe/London"

"America/New_York"

"Asia/Tokyo"

"Europe/Moscow"

"Africa/Monrovia"

"America/Los_Angeles"

"Australia/Sydney"

Google

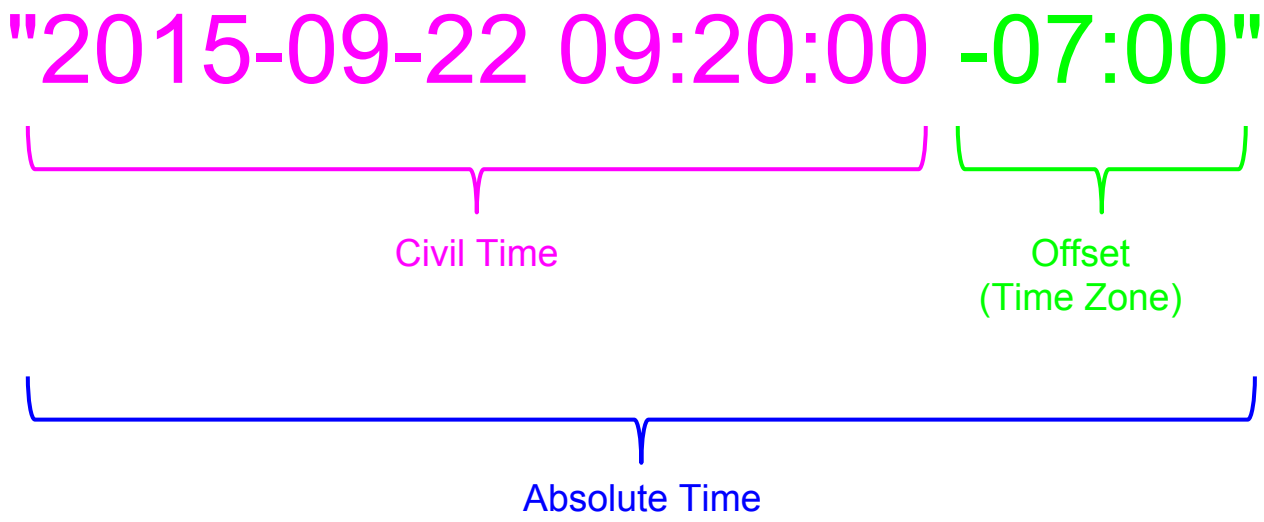
Interesting Time Zone Transitions

- Arizona/Phoenix has no DST
- Asia/Kathmandu jumped 15 min in 1986 (non-DST)
- Australia/Lord_Howe uses a 30 min DST offset
- Pacific/Apia skipped December 30, 2011 (non-DST)
- Africa/Cairo skipped midnight hour in Spring DST transition
- Africa/Monrovia used to use an offset of 44 min. 30 sec.

Pro Tip

Do **not** special-case time transitions!

Diagram of a Time String



Classic Time Programming

Classic Time APIs

- `std::time_t` — non-leap seconds since January 1, 1970 00:00:00 UTC
- `std::tm` — broken-down year, month, day, hour, minute, second, etc.

	UTC	Local Time Zone
`time_t` ⟶ `tm`	`gmtime()`	`localtime()`
`tm` ⟶ `time_t`	X	`mktime()`

Classic Example

```
std::string Format(const std::string& fmt, const std::tm& tm);

int main() {
  const std::time_t now = std::time(nullptr);

  std::tm tm_utc;
  gmtime_r(&now, &tm_utc);
  std::cout << Format("UTC: %F %T\n", tm_utc);

  std::tm tm_local;
  localtime_r(&now, &tm_local);
  std::cout << Format("Local: %F %T\n", tm_local);
}
```

Epoch Shifting

```
int GetOffset(std::time_t t, const std::string& zone);
int main() {
  const std::time_t now = std::time(nullptr);

  // Shift epoch: UTC to "local time_t"
  int off = GetOffset(now, "America/New_York");
  const std::time_t now_nyc = now + off;
  std::tm tm_nyc;
  gmtime_r(&now_nyc, &tm_nyc);
  std::cout << Format("NYC: %F %T\n", tm_nyc);

  // Shift back: "local time_t" to UTC
  off = GetOffset(now_nyc, "America/New_York");
  const std::time_t now_utc = now_nyc - off;
  return now_utc == now ? 0 : 1;
}
```

Epoch Shifting

```
int GetOffset(std::time_t t, const std::string& zone);
int main() {
  const std::time_t now = std::time(nullptr);

  // Shift epoch: UTC to "local time_t"
  int off = GetOffset(now, "America/New_York");
  const std::time_t now_nyc = now + off;
  std::tm tm_nyc;
  gmtime_r(&now_nyc, &tm_nyc);
  std::cout << Format("NYC: %F %T\n", tm_nyc);

  // Shift back: "local time_t" to UTC
  off = GetOffset(now_nyc, "America/New_York");
  const std::time_t now_utc = now_nyc - off;
  return now_utc == now ? 0 : 1;
}
```

Problems:

- `now_nyc` is not really a `time_t`
- What is the offset? Add/sub?
- `std::tm` has some invalid fields
- Local to UTC doesn't work

Pro Tip

No such thing as a "local time_t"

A Simplified Mental Model

F(**Absolute**, TZ) → **Civil**

F(**Civil**, TZ) → **Absolute**

What's missing?

- Time zones are **opaque**
- Numeric offsets unnecessary
- No "local seconds" (i.e., no epoch shifting)
- No access to future/past transitions

"These concepts fill a much needed gap."
— Ken Thompson

Classic APIs in this Model

std::time_t
std::chrono::
system_clock::time_point

UTC or localtime

`std::tm, int`

F(**Absolute**, TZ) → **Civil** `localtime_r()`

F(**Civil**, TZ) → **Absolute** `mktime()`

Announcing: The CCTZ Library NEW

`cctz::time_point`

`cctz::TimeZone`

`cctz::Breakdown`, `int`, etc.

F(**Absolute**, TZ) → **Civil** `cctz::BreakTime()`

F(**Civil**, TZ) → **Absolute** `cctz::MakeTime()`

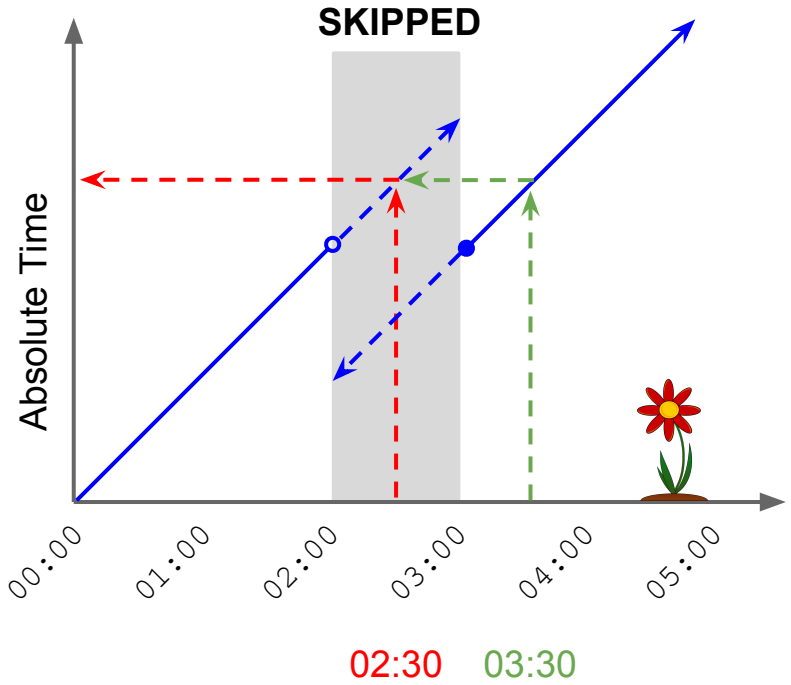
Hello, CCTZ

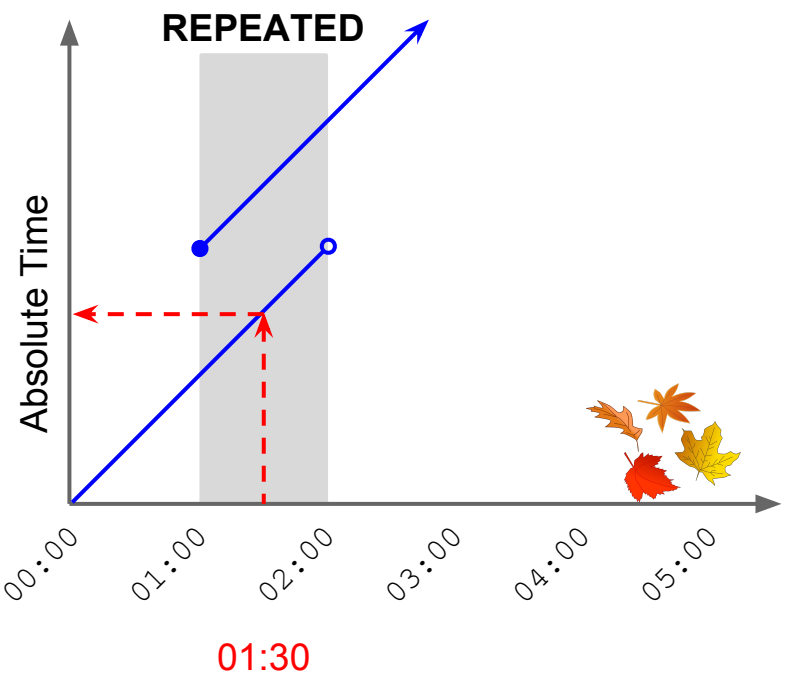
```
int main() {
  cctz::TimeZone syd;
  if (!cctz::LoadTimeZone("Australia/Sydney", &syd)) return -1;

  // Neil Armstrong first walks on the moon
  const cctz::time_point tp1 = cctz::MakeTime(1969, 7, 21, 12, 56, 0, syd);

  const std::string s = cctz::Format("%F %T %z", tp1, syd);
  std::cout << s << "\n";

  cctz::TimeZone nyc;
  cctz::LoadTimeZone("America/New_York", &nyc);

  const cctz::time_point tp2 = cctz::MakeTime(1969, 7, 20, 22, 56, 0, nyc);
  assert(tp2 == tp1);
}
```

The CCTZ Library Implementation

- Posix conventions: Ignores leap seconds, Proleptic Gregorian Calendar
- Uses IANA tzdata from the local system (e.g., /usr/share/zoneinfo)
- Normalizes out-of-range fields (e.g., Oct 32 → Nov 1)
- Faster than libc equivalents
- Nice default handling of discontinuities (e.g., DST)

"Daylight-Saving Time" Transitions





cctz::MakeTime

F(**Civil**, TZ) → **Absolute**

```
struct TimeInfo {
  enum class Kind {
    UNIQUE,    // the civil time was singular (pre == trans == post)
    SKIPPED,   // the civil time did not exist
    REPEATED,  // the civil time was ambiguous
  } kind;
  time_point pre;   // Uses pre-transition offset
  time_point trans;
  time_point post;  // Uses post-transition offset
  bool normalized;
};
TimeInfo MakeTimeInfo(int64_t y, int m, int d, int hh, int mm, int ss,
                      const TimeZone& tz);
time_point MakeTime(int64_t y, int m, int d, int hh, int mm, int ss,
                    const TimeZone& tz);
```

Pro Tip

Use cctz::MakeTime()

cctz::BreakTime

F(**Absolute**, TZ) → **Civil**

```
struct Breakdown {
  int64_t year;          // year (e.g., 2013)
  int month;             // month of year [1:12]
  int day;               // day of month [1:31]
  int hour;              // hour of day [0:23]
  int minute;            // minute of hour [0:59]
  int second;            // second of minute [0:59]
  duration subsecond;    // [0s:1s)
  int weekday;           // 1==Mon, ..., 7=Sun
  int yearday;           // day of year [1:366]
  int offset;            // seconds east of UTC
  bool is_dst;           // is offset non-standard?
  std::string abbr;      // time-zone abbreviation (e.g., "PST")
};
Breakdown BreakTime(const time_point& tp, const TimeZone& tz);
```

Pro Tip

Probably ignore:
offset, is_dst, abbr

Examples

1. Translating Civil Time
2. Parsing
3. Adding Months
4. Calculate "Midnight"
5. Refactor: Epoch Shift

Example 1: Translating Civil Time

```
int main() {
  cctz::TimeZone lax;
  LoadTimeZone("America/Los_Angeles", &lax);

  // Time Programming Fundamentals @cppcon
  const cctz::time_point tp = cctz::MakeTime(2015, 9, 22, 9, 0, 0, lax);

  cctz::TimeZone nyc;
  LoadTimeZone("America/New_York", &nyc);

  std::cout << cctz::Format("Talk starts at %T %z (%Z)\n", tp, lax);
  std::cout << cctz::Format("Talk starts at %T %z (%Z)\n", tp, nyc);
}
```

```
Talk starts at 09:00:00 -0700 (PDT)
Talk starts at 12:00:00 -0400 (EDT)
```

Example 2: Parsing

```
int main() {
  const std::string civil_string = "2015-09-22 09:35:00";

  cctz::TimeZone lax;
  LoadTimeZone("America/Los_Angeles", &lax);
  cctz::time_point tp;
  const bool ok = cctz::Parse("%Y-%m-%d %H:%M:%S", civil_string, lax, &tp);
  if (!ok) return -1;

  const auto now = std::chrono::system_clock::now();
  const std::string s = now > tp ? "running long!" : "on time!";
  std::cout << "Talk " << s << "\n";
}
```

```
Talk on time!
```

Example 3: Adding Months

```
int main() {
  cctz::TimeZone lax;
  LoadTimeZone("America/Los_Angeles", &lax);

  const auto now = std::chrono::system_clock::now();
  const cctz::Breakdown bd = cctz::BreakTime(now, lax);

  // First day of month, 6 months from now.
  const cctz::time_point then =
      cctz::MakeTime(bd.year, bd.month + 6, 1, 0, 0, 0, lax);

  std::cout << cctz::Format("Now: %F %T %z\n", now, lax);
  std::cout << cctz::Format("6mo: %F %T %z\n", then, lax);
}
```

```
Now: 2015-09-17 09:02:41 -0700
6mo: 2016-03-01 00:00:00 -0800
```

Example 4: Floor to "Midnight"

```
cctz::time_point FloorDay(cctz::time_point tp, cctz::TimeZone tz) {
  const cctz::Breakdown bd = cctz::BreakTime(tp, tz);
  const cctz::TimeInfo ti =
      cctz::MakeTimeInfo(bd.year, bd.month, bd.day, 0, 0, 0, tz);
  if (ti.kind == cctz::TimeInfo::Kind::SKIPPED) return ti.trans;
  return ti.pre;
}

int main() {
  cctz::TimeZone lax;
  LoadTimeZone("America/Los_Angeles", &lax);
  const auto now = std::chrono::system_clock::now();
  const auto day = FloorDay(now, lax);
  std::cout << cctz::Format("Now: %F %T %z\n", now, lax);
  std::cout << cctz::Format("Day: %F %T %z\n", day, lax);
}
```

```
Now: 2015-09-17 09:12:53 -0700
Day: 2015-09-17 00:00:00 -0700
```

Refactor: Old Code

```
void GetLocalTime(std::time_t t, const std::string& zone,
                  int* hour, int* min, int* sec) {

                    ...


}
```

Refactor: Old Code (Epoch Shifting)

```
void GetLocalTime(std::time_t t, const std::string& zone,
                  int* hour, int* min, int* sec) {
  int off;
  CalcOffset(zone, t, &off);
  std::time_t adjusted_time = t + off;
  std::tm gm_tm;
  ::gmtime_r(&adjusted_time, &gm_tm);
  *hour = gm_tm.tm_hour;
  *min = gm_tm.tm_min;
  *sec = gm_tm.tm_sec;
}
```

Refactor: NEW Code

```
void GetLocalTime(std::time_t t, const std::string& zone,
                  int* hour, int* min, int* sec) {
  cctz::TimeZone tz;
  cctz::LoadTimeZone(zone, &tz);
  const cctz::time_point tp = std::chrono::system_clock::from_time_t(t);
  const cctz::Breakdown bd = cctz::BreakTime(tp, tz);
  *hour = bd.hour;
  *min = bd.minute;
  *sec = bd.second;
}
```

Refactor: NEW Code — BETTER

```
void GetLocalTime(cctz::time_point tp, cctz::TimeZone tz,
                  int* hour, int* min, int* sec) {
  const cctz::Breakdown bd = cctz::BreakTime(tp, tz);
  *hour = bd.hour;
  *min = bd.minute;
  *sec = bd.second;
}
```

Refactor: NEW Code — BEST

```
void GetLocalTime(cctz::time_point tp, cctz::TimeZone tz,
                  int* hour, int* min, int* sec) {
  const cctz::Breakdown bd = cctz::BreakTime(tp, tz);
  *hour = bd.hour;
  *min = bd.minute;
  *sec = bd.second;
}
```

Pro Tip

Directly call:
cctz::BreakTime(tp, tz)

Final Thoughts

Pro Tips

- Use the mental model
- Use proper vocabulary
- Use "%z" in format strings — produces an **absolute time**
- Never compute with UTC offsets (no `time_t` math, no epoch shifting, etc)
- Compute `cctz::Breakdown` rather than pass it
- Do calendar-like calculations in the **civil time** domain
- Terminology: UTC rather than GMT

CCTZ Available Today

github.com/google/cctz

Q&A

github.com/google/cctz

Greg Miller (jgm@google.com)
Bradley White (bww@google.com)

Google