The C++ Time Class

The `Time` class provides basic facilities for getting the current time, constructing time intervals, adding and subtracting times, and comparing times:

C++

```
namespace IceUtil {

    typedef ... Int64;

    class Time {
    public:
        enum Clock { Realtime, Monotonic };
        Time(Clock = Realtime);
        static Time now();
        static Time seconds(Int64);
        static Time milliSeconds(Int64);
        static Time microSeconds(Int64);

        Int64 toSeconds() const;
        Int64 toMilliSeconds() const;
        Int64 toMicroSeconds() const;

        double toSecondsDouble() const;
        double toMilliSecondsDouble() const;
        double toMicroSecondsDouble() const;

        std::string toDateTime() const;
        std::string toDuration() const;

        Time operator-() const;

        Time operator-(const Time&) const;
        Time operator+(const Time&) const;

        Time operator*(int) const;
        Time operator*(Int64) const;
        Time operator*(double) const;

        double operator/(const Time&) const;
        Time operator/(int) const;
        Time operator/(Int64) const;
        Time operator/(double) const;

        Time& operator-=(const Time&);
        Time& operator+=(const Time&);

        Time& operator*=(int);
        Time& operator*=(Int64);
        Time& operator*=(double);

        Time& operator/=(int);
        Time& operator/=(Int64);
        Time& operator/=(double);

        bool operator<(const Time&) const;
        bool operator<=(const Time&) const;
        bool operator>(const Time&) const;
        bool operator>=(const Time&) const;
        bool operator==(const Time&) const;
        bool operator!=(const Time&) const;

#ifndef _WIN32
        operator timeval() const;
#endif
    };

    std::ostream& operator<<(std::ostream&, const Time&);
}
```

The member functions behave as follows:

Time

Internally, the `Time` class stores ticks in microsecond units. For absolute time, this is the number of microseconds since the Unix epoch (00:00:00 UTC on 1 Jan. 1970). For durations, this is the number of microseconds in the duration. The default constructor initializes the tick count to zero and selects the real-time clock. Constructing `Time` with an argument of `Monotonic` selects the monotonic clock on platforms that support it; the real-time clock is used on other platforms.

now

This function constructs a `Time` object that is initialized to the current time of day.

seconds, milliSeconds, microSeconds

These functions construct `Time` objects from the argument in the specified units. For example, the following statement creates a time duration of one minute:

C++

```
IceUtil::Time t = IceUtil::Time::seconds(60);
```

toSeconds, toMilliSeconds, toMicroSeconds

The member functions provide explicit conversion of a duration to seconds, milliseconds, and microseconds, respectively. The return value is a 64-bit signed integer (`IceUtil::Int64`). For example:

C++

```
IceUtil::Time t = IceUtil::Time::milliSeconds(2000);
IceUtil::Int64 secs = t.toSeconds(); // Returns 2
```

toSecondsDouble, toMilliSecondsDouble, toMicroSecondsDouble

The member functions provide explicit conversion of a duration to seconds, milliseconds, and microseconds, respectively. The return value is of type `double`.

toDateTime

This function returns a human-readable representation of a `Time` value as a date and time.

toDuration

This function returns a human-readable representation of a `Time` value as a duration.

Operators

`Time` provides operators that allow you to add, subtract, multiply, and divide times. For example:

C++

```
IceUtil::Time oneMinute = IceUtil::Time::seconds(60);
IceUtil::Time oneMinuteAgo = IceUtil::Time::now() - oneMinute;
```

The multiplication and division operators permit you to multiply and divide a duration. Note that these operators provide overloads for `int`, `long long`, and `double`.

The comparison operators allow you to compare times and time intervals with each other, for example:

```
IceUtil::Time oneMinute  = IceUtil::Time::seconds(60);
IceUtil::Time twoMinutes = IceUtil::Time::seconds(120);
assert(oneMinute < twoMinutes);
```

The `timeval` operator converts a `Time` object to a `struct timeval`, defined as follows:

C++

```
struct timeval {
    long tv_sec;
    long tv_usec;
};
```

The conversion is useful for API calls that require a `struct timeval` argument, such as `select`. To convert a duration into a `timeval` structure, simply assign a `Time` object to a `struct timeval`:

C++

```
IceUtil::Time oneMinute = IceUtil::Time::seconds(60);
struct timeval tv;
tv = t;
```

Note that this member function is not available under Windows.

```
std::ostream& operator<<(std::ostream&, Time&);
```

This operator prints the number of whole seconds since the epoch.

See Also

- The C++ Timer and TimerTask Classes