Ruby trunk - Feature #13919

Add a new method to create Time instances from unix time and nsec

09/19/2017 06:59 AM - tagomoris (Satoshi TAGOMORI)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

Time object contains nsec, but Time class doesn't have method to create an instance with nsec.
Time.at() accepts 2nd argument, but it's micro-sec, and we need to divide nsec by 1000.0.

```
t1 = Time.now
t1.nsec #=> nsec value

t2 = Time.at(t1.to_i, t1.usec) # not nsec!
t3 = Time.at(t1.to_i, t1.nsec / 1000.0) # additional division
```

I think it's better to have another method to create an instance from time(unix time) and nsec, for example, Time.of(unix_time, nsec)

```
t = Time.now
Time.of(t.to_i, t.nsec) == t
```

Associated revisions

Revision 97c5a33f - 09/25/2017 06:20 AM - naruse (Yui NARUSE)

Time#at receives 3rd argument which specifies the unit of 2nd argument [Feature #13919]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@60017 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 60017 - 09/25/2017 06:20 AM - naruse (Yui NARUSE)

Time#at receives 3rd argument which specifies the unit of 2nd argument [Feature #13919]

Revision 60017 - 09/25/2017 06:20 AM - naruse (Yui NARUSE)

Time#at receives 3rd argument which specifies the unit of 2nd argument [Feature #13919]

Revision 60017 - 09/25/2017 06:20 AM - naruse (Yui NARUSE)

Time#at receives 3rd argument which specifies the unit of 2nd argument [Feature #13919]

History

#1 - 09/19/2017 11:49 PM - shevegen (Robert A. Heiler)

I have no objection to a new method at all.

I think the name "Time.of()" is strange though.

With Time.at(), we can say "ok, at this or that moment, we want the time".

With Time.of(), I am confused what that should mean. I am not a native
english speaker though, so who knows.

Would Time.nsec be any worse? That is, it would want the time input in
nanoseconds (nsec is nano seconds right?)

I agree that Time.at() is not very convenient with the second parameter
mandating microsecond IF one wants to use nanoseconds instead. I think
the reason why that was chosen, though, was because nsec may be less
frequent than msec. Like hour-minutes-seconds notation, and then the
milliseconds after a ',' or so.

Just putting some more ideas for names out there:

Time.at_nsec

Time.at(t.to_i, nsec: t)
Time.at(t.to_i, :nsec)

Hmm. Or perhaps be able to modify the default for Time so that you can switch it to nsec, like:

Time.default_to = :microseconds
Time.default_to = :nanoseconds

Or anything like that. Just trying to find good alternatives, giving things a good name that can easily stick mentally is not always easy.

#2 - 09/20/2017 06:50 AM - tagomoris (Satoshi TAGOMORI)

Time.at_nsec is ok for me, but it seems to take just one argument of nano seconds from epoch.

- Time.at_nsec(nsec_from_epoch)
- Time.at_nsec(seconds_from_epoch, nsec)

Is it not confusing? Or should it get exactly two arguments?

#3 - 09/20/2017 01:46 PM - akr (Akira Tanaka)

I think new method is possible if we find a good method name.

Another idea is adding an optional argument, unit, for Time.at.
Time.at(s, ns, :nanosecond) for example.

#4 - 09/20/2017 09:31 PM - naruse (Yui NARUSE)

Following is a patch based on akr's API.

```
diff --git a/test/ruby/test_time.rb b/test/ruby/test_time.rb
index c92aafc149..31d245ade7 100644
--- a/test/ruby/test_time.rb
+++ b/test/ruby/test_time.rb
@@ -236,6 +236,17 @@ def test_at2
     assert_equal(1, Time.at(0, 0.001).nsec)
   end

+  def test_at_with_unit
+    assert_equal(123456789, Time.at(0, 123456789, :nanosecond).nsec)
+    assert_equal(123456789, Time.at(0, 123456789, :nsec).nsec)
+    assert_equal(123456000, Time.at(0, 123456, :microsecond).nsec)
+    assert_equal(123456000, Time.at(0, 123456, :usec).nsec)
+    assert_equal(123456000, Time.at(0, 123456, nil).nsec)
+    assert_equal(123000000, Time.at(0, 123, :millisecond).nsec)
+    assert_raise(ArgumentError){ Time.at(0, 1, 2) }
+    assert_raise(ArgumentError){ Time.at(0, 1, :invalid) }
+  end
+
   def test_at_rational
     assert_equal(1, Time.at(Rational(1,1) / 1000000000).nsec)
     assert_equal(1, Time.at(1167609600 + Rational(1,1) / 1000000000).nsec)
diff --git a/time.c b/time.c
index e55628fc1b..2431a2406a 100644
--- a/time.c
+++ b/time.c
@@ -36,6 +36,7 @@

 static ID id_divmod, id_submicro, id_nano_num, id_nano_den, id_offset, id_zone;
 static ID id_quo, id_div;
+static ID id_nanosecond, id_microsecond, id_millisecond, id_nsec, id_usec;

 #define NDIV(x,y) (-(-((x)+1)/(y))-1)
 #define NMOD(x,y) ((y)-(-((x)+1)%(y))-1)
@@ -2347,11 +2348,32 @@ time_s_now(VALUE klass)
     return rb_class_new_instance(0, NULL, klass);
 }

+static int
+get_scale(VALUE unit) {
+    if (unit == ID2SYM(id_nanosecond) || unit == ID2SYM(id_nsec)) {
+        return 1000000000;
+    }
```

```
+    else if (NIL_P(unit) || unit == ID2SYM(id_microsecond) || unit == ID2SYM(id_usec)) {
+        return 1000000;
+    }
+    else if (unit == ID2SYM(id_millisecond)) {
+        return 1000;
+    }
+    else {
+        rb_raise(rb_eArgError, "unexpected unit: %"PRIsVALUE, unit);
+    }
+}
+
 /*
  *  call-seq:
  *     Time.at(time) -> time
  *     Time.at(seconds_with_frac) -> time
  *     Time.at(seconds, microseconds_with_frac) -> time
+ *     Time.at(seconds, milliseconds, :millisecond) -> time
+ *     Time.at(seconds, microseconds, :usec) -> time
+ *     Time.at(seconds, microseconds, :microsecond) -> time
+ *     Time.at(seconds, nanoseconds, :nsec) -> time
+ *     Time.at(seconds, nanoseconds, :nanosecond) -> time
  *
  *  Creates a new Time object with the value given by +time+,
  *  the given number of +seconds_with_frac+, or
@@ -2362,24 +2384,26 @@ time_s_now(VALUE klass)
  *
  *  If a numeric argument is given, the result is in local time.
  *
-*     Time.at(0)                            #=> 1969-12-31 18:00:00 -0600
-*     Time.at(Time.at(0))                   #=> 1969-12-31 18:00:00 -0600
-*     Time.at(946702800)                    #=> 1999-12-31 23:00:00 -0600
-*     Time.at(-284061600)                   #=> 1960-12-31 00:00:00 -0600
-*     Time.at(946684800.2).usec             #=> 200000
-*     Time.at(946684800, 123456.789).nsec  #=> 123456789
+ *     Time.at(0)                                 #=> 1969-12-31 18:00:00 -0600
+ *     Time.at(Time.at(0))                        #=> 1969-12-31 18:00:00 -0600
+ *     Time.at(946702800)                         #=> 1999-12-31 23:00:00 -0600
+ *     Time.at(-284061600)                        #=> 1960-12-31 00:00:00 -0600
+ *     Time.at(946684800.2).usec                  #=> 200000
+ *     Time.at(946684800, 123456.789).nsec        #=> 123456789
+ *     Time.at(946684800, 123456789, nsec).nsec   #=> 123456789
  */

 static VALUE
 time_s_at(int argc, VALUE *argv, VALUE klass)
 {
-    VALUE time, t;
+    VALUE time, t, unit = Qnil;
     wideval_t timew;

-    if (rb_scan_args(argc, argv, "11", &time, &t) == 2) {
+    if (rb_scan_args(argc, argv, "12", &time, &t, &unit) >= 2) {
+        int scale = get_scale(unit);
         time = num_exact(time);
         t = num_exact(t);
-        timew = wadd(rb_time_magnify(v2w(time)), wmulquoll(v2w(t), TIME_SCALE, 1000000));
+        timew = wadd(rb_time_magnify(v2w(time)), wmulquoll(v2w(t), TIME_SCALE, scale));
         t = time_new_timew(klass, timew);
     }
     else if (IsTimeval(time)) {
@@ -4819,6 +4843,11 @@ Init_Time(void)
     id_nano_den = rb_intern("nano_den");
     id_offset = rb_intern("offset");
     id_zone = rb_intern("zone");
+    id_nanosecond = rb_intern("nanosecond");
+    id_microsecond = rb_intern("microsecond");
+    id_millisecond = rb_intern("millisecond");
+    id_nsec = rb_intern("nsec");
+    id_usec = rb_intern("usec");

     rb_cTime = rb_define_class("Time", rb_cObject);
     rb_include_module(rb_cTime, rb_mComparable);
```

#5 - 09/25/2017 05:51 AM - matz (Yukihiro Matsumoto)

I agree with the new behavior (except for accepting nil as a unit).

Matz.

#6 - 09/25/2017 06:20 AM - naruse (Yui NARUSE)

- Status changed from Open to Closed

Applied in changeset trunk|r60017.

Time#at receives 3rd argument which specifies the unit of 2nd argument [Feature #13919]