Ruby - Bug #21301

Invalid Dates Accepted When Using "UTC" in Time.new

05/02/2025 09:32 PM - mame (Yusuke Endoh)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	3.2: UNKNOWN, 3.3: UNKNOWN, 3.4: UNKNOWN

Description

2025-04-31 does not exist, but, when creating a time object with the "UTC" zone, the value is accepted without error:

```
Time.new(2025, 4, 31, 0, 0, 0, "UTC") #=> expected: 2025-05-01 00:00:00 UTC
                                      #=> actual:   2025-04-31 00:00:00 UTC
```

In contrast, using the "+00:00" time zone works as expected and rolls the date over to 2025-05-01:

```
Time.new(2025, 4, 31, 0, 0, 0, "+00:00") # => 2025-05-01 00:00:00 +0000
```

Note that 2025-04-30T24:00:00Z is correctly rolled over to 2025-05-01T00:00:00Z.
And 2025-04-31T24:00:00Z is rolled over to 2025-04-01!

```
Time.new(2025, 4, 30, 24, 0, 0, "UTC") #=> 2025-05-01 00:00:00 UTC # OK
Time.new(2025, 4, 31, 24, 0, 0, "UTC") #=> 2025-04-01 00:00:00 UTC # What?
```

Related issues:

Related to Ruby - Feature #21307: A way to strictly validate time input	**Assigned**

History

#1 - 05/03/2025 03:35 AM - dodecadaniel (Daniel Colson)

Possible fix: https://github.com/ruby/ruby/pull/13246

#2 - 05/03/2025 04:34 PM - mame (Yusuke Endoh)

Thanks. Actually, the same issue also occurs with a value of 60 for seconds:

```
Time.new(2025, 1, 1, 0, 0, 60, "UTC")    #=> 2025-01-01 00:00:60 UTC   # invalid
Time.new(2025, 1, 1, 0, 0, 60, "+00:00") #=> 2025-01-01 00:01:00 +0000
```

Note that 60 seconds can be valid in the case of a leap second.

```
Time.new(2016, 12, 31, 23, 59, 60, "UTC")    #=> 2016-12-31 23:59:60 UTC   # can be valid
Time.new(2016, 12, 31, 23, 59, 60, "+00:00") #=> 2016-12-31 23:59:60 +0000
```

2016-12-31 23:59:60 UTC is valid if the system has leap second data.

#3 - 05/04/2025 02:05 AM - dodecadaniel (Daniel Colson)

Ah yeah, rolling over seconds seems trickier. I think the time with the offset doesn't have to deal with all that because it doesn't set the vtm struct right away. Instead, it only sets the timew (a kind of timestamp) and then calculates the vtm later as needed based on the timew. The vtm -> timew -> vtm round trip is enough to normalize everything (that uses timegmw and gmtimew, which are leap-second aware, etc.).

The UTC time tries to put together and set the vtm struct directly upfront and then sets tobj->vtm.tm_got = 1. Changing tm_got from 1 to 0 is enough to get the UTC code behaving like the offset code, but I assume there's some cost to that. Possibly worth it for better accuracy though? I can explore that more if it seems worthwhile. It'd save us from needing to do vtm_day_wraparound too.

#4 - 05/07/2025 05:04 AM - mame (Yusuke Endoh)

- Related to Feature #21307: A way to strictly validate time input added