Declare a TDateTime as a Const in Delphi

Asked 15 years, 9 months ago Modified 4 years, 3 months ago Viewed 13k times

31

As far as I know there is no way to do this, but I am going to ask just in case someone else knows how to do this. How can I declare a date as a const in Delphi?

The only solution I have found is to use the numeric equivalent, which is **kind of a pain** to maintain because it is not human readable.

```
const
  Expire : TDateTime = 39895; // Is actually 3/23/2009
```

What I would like to be able to do is something like this:

```
const
  Expire : TDateTime = TDateTime ('3/23/2009');
```

or

```
const
  Expire : TDateTime = StrToDate('3/23/2009');
```

So let me know if this is a feature request or if I just missed how to do this (yeah, I know it seems like an odd thing to want . . . .)

delphi date operator-overloading constants tdatetime

Share

Improve this question

Follow

edited May 15, 2009 at 17:16

Wouter van Nifterick

24.1k • 7 • 81 • 123

asked Mar 23, 2009 at 23:00

Jim McKeeth

38.7k • 25 • 124 • 199

Great question - I've often wanted to do this (more for times than for dates, but the principle is much the same - eg I want to put 6:45pm into a TDateTime as a const, etc). I invariably end up doing something like your first example, with comments - and it's a pain when I later need to change it! – robsoft Mar 24, 2009 at 5:38

This looks like something GExperts or DLangExtensions should be able to do, either as an expert to enter date and or time to create a properly commented constant, or as a preprocessor converting string to TDateTime. Allowing ISO 8601 formats only should remove all ambiguity. – mghie Mar 24, 2009 at 5:59

@Mghie - good point. A GExperts solution would be perfectly acceptable to me. It's when I have to get the calculator out to start doing divisions that I get frustrated. :-) – robsoft Mar 24, 2009 at 6:13

11 Answers

Sorted by: Highest score (default)

23

Ok, my reaction is a bit late, but here's a solution for the newer Delphi's.

It uses implicit class overloaders so that records of this type can be used as if they are TDateTime variables.

```
TDateRec = record
  year,month,day,hour,minute,second,millisecond:word;
  class operator implicit(aDateRec:TDateRec):TDateTime;
  class operator implicit(aDateTime:TDateTime):TDateRec; // not needed
  class operator implicit(aDateRec:TDateRec):String; // not needed
  class operator implicit(aDateRec:String):TDateRec; // not needed
end;
```

Implementation:

```
uses DateUtils;

class operator TDateRec.Implicit(aDateRec:TDateRec):TDateTime;
begin
  with aDateRec do // Yeah that's right you wankers. I like "with" :)
    Result := encodeDateTime(Year,Month,Day,Hour,Minute,Second,Millisecond);
end;

class operator TDateRec.Implicit(aDateTime:TDateTime):TDateRec;
begin
  with Result do
    DecodeDateTime(aDateTime,Year,Month,Day,Hour,Minute,Second,Millisecond);
end;

class operator TDateRec.Implicit(aDateRec:TDateRec):String;
begin
  Result := DateTimeToStr(aDateRec)
end;

class operator TDateRec.Implicit(aDateRec:String):TDateRec;
begin
  Result := StrToDateTime(aDateRec)
end;
```

Now you can declare your dates like this:

```
const
  Date1:TDateRec=(Year:2009;month:05;day:11);
```

```
  Date2:TDateRec=(Year:2009;month:05;day:11;hour:05);
  Date3:TDateRec=(Year:2009;month:05;day:11;hour:05;minute:00);
```

To see if it works, execute the following:

```
ShowMessage(Date1); // it can act like a string
ShowMessage(DateToStr(Date1)); // it can act like a date
```

If you really want to replace all your TdateTime variables with this, you probably need to overload some other operators too (Add, subtract, explicit, ...).

Share

Improve this answer

Follow

edited May 15, 2009 at 17:09

answered May 15, 2009 at 17:03

Wouter van Nifterick

24.1k ● 7 ● 81 ● 123

2 +1 for the solution and +1 again for `with`, although I despise its use :-) – Jerry Dodge Feb 26, 2014 at 0:26

I took this and added my own additions to it... `FriendlyDate`, `FriendlyTime`, and `FriendlyDateTime` – Jerry Dodge Feb 26, 2014 at 0:55

12

The only? possible way, but probably not what you are looking for:

```
const
{$J+}
  Expire: TDateTime = 0;
{$J-}

initialization
  Expire := EncodeDate(2009, 3, 23);
```

Share

Improve this answer

Follow

edited Mar 24, 2009 at 19:30

Jim McKeeth
38.7k • 25 • 124 • 199

answered Mar 23, 2009 at 23:26

The_Fox
7,022 • 2 • 46 • 70

2 May as well just declare it as a var instead of a const. – Rob Kennedy Mar 24, 2009 at 0:30

1 Yes, you are right, but at least my solution uses the const keyword ;) – The_Fox Mar 24, 2009 at 7:44

Jim, my typo originates from your post :P – The_Fox Mar 24, 2009 at 20:40

Ooops, you are right. The first Expire was spelled wrong. I will fix it. I thought it might when I saw it, but then I only looked at the 2nd two instances. – Jim McKeeth Mar 24, 2009 at 21:35

1 Please don't ever do this. I hate {$J+} – Warren P Apr 9, 2015 at 14:48

11

I tend to simulate **const** dates with a function. Technically they're a little more *constant* than the "pseudo-constant" assignable typed **const**'s.

```
function Expire: TDateTime;
begin
  Result := EncodeDate(2009, 3, 23);
end;
```

NOTE the use of `EncodeDate` rather than `StrToDate`. `StrToDate` is affected by regional settings meaning there's no guarantee a string will be interpreted as would be expected.

For example, did you know that there's a strange a group of people who think it makes sense to "shuffle" date parts into an inconsistent order of significance? They use middle, then least, then most significant part (e.g. '3/23/2009') **<cheeky grin>**. The only time that logic makes sense is when you turn 102 years old - then you can claim your age is 021.

> For the *premature optimisers* out there, if the function is called so frequently that the nano seconds required to encode a date becomes an issue - you have a far **bigger** problem than this minor inefficiency in the name of readable, maintainable code.

Share Improve this answer Follow

answered Jun 30, 2012 at 12:31
Disillusioned
14.8k 3 47 80

8

There is no way to do this because interpreting a date litteral in itself is not deterministic, it depends on the convention/locale you follow.
'1/4/2009' is not in January for any French person for instance, and having the compiler translating as January 4th would make it a fool's compiler ;-)
Unless the compiler implements some (well documented) "magic" bijective function for pairing a date value and a display representation... And anyway, half of the planet would not like it.
The only non ambiguous way I see now is to provide the value even if it looks like a pain... ... my $0.02

Share Improve this answer Follow

answered Mar 24, 2009 at 1:30
Francesca
21.6k 5 52 91

3 It only depends on the locale if the language is defined to be sensitive to the locale. Floating-point literals aren't locale-sensitive; they simply don't accept commas as the decimal separator. No reason a date-time literal couldn't have similar restrictions on format. – Rob Kennedy Mar 24, 2009 at 2:08

1 EncodeDate and EncodeTime are deterministic so would be useable **if** they could be evaluated at compile time. – Disillusioned Jun 6, 2012 at 15:14

6

No, Delphi doesn't support that.

Your first idea would be a request for date-time literals distinct from ordinary floating-point literals. I found QC 72000, which is about displaying `TDateTime` values as dates in the debugger, but nothing about your particular request. It's not like nobody's ever mentioned it before, though. It's a perennial topic on the newsgroups; I just can't find anything in QC about it.

Your second idea would require `StrToDate` to be evaluable at compile time. I don't see any entries in QC about it either, but for what it's worth, C++ is getting such a feature for functions that are shown to have the necessary qualities. `StrToDate`

wouldn't meet those requirements, though, because it's sensitive to the current locale's date settings.

Share Improve this answer Follow

answered Mar 24, 2009 at 1:10

Rob Kennedy
163k • 23 • 284 • 477

1 EncodeDate, EncodeTime aren't affected and would suit the requirement. – Disillusioned Jun 6, 2012 at 15:13

Not quite, @Craig. One of C++'s requirements is that the function cannot throw exceptions (implicit in the requirement that the function consist solely of a `return` statement). The Delphi functions have compound statements and can throw. – Rob Kennedy Jun 6, 2012 at 16:30

1 A slight misunderstanding, I should have been less ambiguous in my comment. Allow me to elaborate: EncodeDate and EncodeTime aren't affected (by localisation - they're deterministic) and would suit the requirement (of Jim to specify a constant date/time, albeit syntactically different to his question). Of course the idea is a moot point since Delphi currently doesn't have such a feature. :(However, your comment about C++'s no exception requirement is interesting. – Disillusioned Jun 6, 2012 at 16:51

4

Rob Kennedy's answer shows that the StrToDate solution is inherently out of the question as you don't want your code to break if it's compiled in Europe!

I do agree there should be some way to do EncodeDate but there isn't.

As far as I'm concerned the compiler should simply compile and run any code it finds in a constant assignment and store the result into the constant. I'd leave it up to the programmer to ensure the code isn't sensitive to it's environment.

Share Improve this answer Follow

answered Mar 24, 2009 at 1:27

Loren Pechtel
9,073 • 4 • 35 • 46

4

One solution would be to create a list of constants for years, another for month offsets and then build it on the fly. You would have to take care of leap years yourself by adding 1 to each resulting constant. Just a few below to get you started... :)

```
Const
  Leap_Day = 1;  // use for clarity for leap year dates beyond feb 29.
  Year_2009 = 39812;  // January 1, 2009
  Year_2010 = Year_2009 + 365; // January 1, 2010
  Year_2011 = Year_2010 + 365; // January 1, 2011
  Year_2012 = Year_2011 + 365; // January 1, 2012 (is leap year)
  Year_2013 = Year_2012 + Leap_Day + 365;  // January 1, 2013
```

```
Const
  Month_Jan = -1; // because adding the day will make the offset 0.
  Month_Feb = Month_Jan + 31; // 31 days more for the first day of Feb.
  Month_Mar = Month_Feb + 28; // 28 days more for the first day of Mar.
  Month_Apr = Month_Mar + 30; // 30 days more for the first day of Apr.

Const
  Expire_Jan1 : tDateTime = Year_2009 + Month_Jan + 1;
  Expire : tDateTime = Year_2009 + Month_Mar + 23;
```

If you have a leap year then you have to add 1 to anything beyond february of that year.

```
Const
  Expire : tDateTime = Year_2008 + Month_Mar + 23 + Leap_Day;
```

EDIT

Added a few more years for clarity, and added a Leap_Day constant.

Share

Improve this answer

Follow

edited Mar 26, 2009 at 16:08

answered Mar 25, 2009 at 16:49

skamradt

15.5k 3 39 57

Why not add another constant: Leap_day = 1. That way there's no spurious "+1" sitting around that may not be intuitive at first glance... – Mason Wheeler Mar 25, 2009 at 20:57

3

A Delphi date is the # of days since Dec 30, 1899. So you could probably come up with an elaborate mathematical formula to express a date as a const. Then you could format it very oddly, to emphasize the human-readable parts. My best attempt is below, but it is very much incomplete; for one thing, it assumes that all months have 30 days.

My example is mostly for fun though. In practice, this is pretty ridiculous.

```
const
  MyDate = ((
            2009   //YEAR
                                                - 1900) * 365.25) + ((
            3      //MONTH
                                                - 1) * 30) +
            24     //DAY
            ;
```

Share Improve this answer Follow

answered Mar 24, 2009 at 13:36

JosephStyons

58.6k 64 167 237

the fractional part also can cause problems as your date slowly moves from midnight one year, to 6 am, noon, 6pm to return again on leap year to midnight. – skamradt Mar 25, 2009 at 17:07

Yeah, absolutely true. Like I said, this is not really workable unless you spend a lot of time devising a really sophisticated forumla to handle all possibilities. Much better off just getting the value and hard-coding it. – JosephStyons Mar 25, 2009 at 17:43

And you may have a very hard-to-find bug if you accidentally get a digit or operator or bracket wrong in the "margin". Not going to downvote, but please don't do this in real code. – Andreas Rejbrand Oct 30, 2019 at 11:12

1

i think *the* best solution available to you is to declare:

```
ArmisticeDay: TDateTime = 6888.0 + (11.0/24.0); //Nov 11, 1918 11 AM
```

and just accept it.

My attempt Nº1

```
Expire = EncodeDate(2009, 3, 23);
```

> [Error] Constant expression expected

My attempt Nº2

```
Expire: TDateTime = EncodeDate(2009, 3, 23);
```

> [Error] Constant expression expected

So even though they're constant, and deterministic (i.e. do not depend on any locale information), it still doesn't work.

Share

Improve this answer

Follow

edited May 21, 2009 at 12:59

answered May 14, 2009 at 17:09

Ian Boyd

256k • 264 • 907 • 1.3k

The type "TDateTime" = type "Double".

1

Algorithm:

1. Use **StrToDateTime('01.01.1900 01:01:01')** *(or other way)* to calculate need double_value. ('01.01.1900 01:01:01' => 2.04237268518519)

2.**const DTZiro: TDateTime = 2.04237268518519;**

Share Improve this answer Follow

answered Nov 18, 2018 at 8:42
jeckp
21 • 1

The OP always knows this. – Andreas Rejbrand Oct 30, 2019 at 11:14

1

System.DateUtils has constants for each part of time.

```
const cDT : TDateTime = (12 * OneHour)   + ( 15 * OneMinute)
                      + (33 * OneSecond) + (123 * OneMillisecond);
```

Share

Improve this answer

Follow

edited Sep 11, 2020 at 0:27
Dominique Fortin
2,238 • 15 • 21

answered Sep 10, 2020 at 14:04
Robert Babiak
11 • 3

That is really creative. I didn't think of that. – Jim McKeeth Sep 12, 2020 at 9:23