

Re: 'Base

Source: <http://coding.derkeiler.com/Archive/Ada/comp.lang.ada/2005-12/msg00109.html>

- *From:* "Randy Brukardt" <randy@xxxxxxxxxxxxxxx>
 - *Date:* Thu, 8 Dec 2005 20:57:49 -0600
-

"Matthew Heaney" <mheaney@xxxxxxx> wrote in message
news:1134072443.635504.296550@xx

>

> Jeffrey R. Carter wrote:

>>

>> I'm surprised ObjectAda uses 32 bits for this. GNAT 3.4.2 uses 8 bits
(-128 .. 127).

>

> Exactly my point. If you assume that you have more range than the RM
> promises, then you're only asking for trouble.

Not quite true. The range is always going to correspond to the representation; so unless you have to worry about non-binary machines, you can assume the next higher power of 2 minus 1; in this case 63. Of course, usually this isn't particularly relevant – the extra values are a small percentage of the total.

Moreover, if you only want to worry about typical machines, then the values are only going to be some multiple of 8-bits, so 'Base'Last can pretty much be assumed to be the smallest value of $2^{(8*N-1)}-1$ without trouble.

But of course the whole point of these attributes is so you don't have to assume at all.

BTW, 'Base works on any scalar type, and the rules for floats and fixed point types are somewhat different than described in the messages here.

And a quick answer for the OP: using 'Base is usually only needed in various tricks. Here's one. Imagine you want a type that can hold at least 0 .. 1000, but you don't care about the exact bounds and want the most efficient type possible. You can declare:

```
type Restricted_Type is range 0 .. 1000;  
type Big_Enough_Type is range Restricted_Type'Base'First ..  
Restricted_Type'Base'Last;
```

and Big_Enough_Type will be some full-range type appropriate for the machine (probably 16-bits or 32-bits).

Re: 'Base

Matt gave another example. Suppose you have a generic:

generic

type G_Int is range <>;

package Gen is

...

and you need a counter of type G_Int that starts at zero and goes to the upper bound of G_Int. Sloppy programmers would just use G_Int for such a counter, but that would fail if the range of G_Int didn't include zero.

Slightly less sloppy programmers would declare the counter as having the type G_Int'Base, but that isn't going to have the overflow check. The best solution is to declare the counter as:

Counter : G_Int'Base range 0 .. G_Int'Last;

which ensures that 0 is included, and the right range is used. Note that this could cover a larger range than the original type.

Randy.

• **References:**

◆ **'Base**

◇ From: ada_student

◆ **Re: 'Base**

◇ From: Matthew Heaney

◆ **Re: 'Base**

◇ From: Martin Dowie

◆ **Re: 'Base**

◇ From: Jeffrey R. Carter

◆ **Re: 'Base**

◇ From: Matthew Heaney

• Prev by Date: **Re: Controlled types and exception safety**

• Next by Date: **Re: Avoiding constraint checks w/ 'Base**

• Previous by thread: **Re: 'Base**

• Next by thread: **Avoiding constraint checks w/ 'Base**

• Index(es):

◆ **Date**

◆ **Thread**