MapShots Unit-of-Measure Library

Aug 19, 2014

MapShots has built a Unit-of-Measure (UOM) engine that we use in all of our applications. It is a self-contained .Net dll with no dependencies. To the best of our knowledge, it can be compiled as a PCL component.

The engine was originally built using enumerations for the compact representation of a simple unit, or a compound unit. A simple unit is “pound”. A compound unit is “pound/acre”. It supports a broad range of simple units and their conversions, and the operations related to simple units that are combined into compound units.

The enumeration was used to find a matching UOM object within an internal list of UOM objects. A UOM object has properties for Name, Abbreviation, PluralName, BaseUnit, Domain, System, and ConversionToBase factor. Using this model, units are always converted within the same system, whenever possible, reducing the typical “quarts to gallons converting through liter” rounding issues.

In addition to the objects that one can access (providing text for reports, pick lists, etc), there is a conversion factory class with static methods for common conversion tasks. You can do simple conversions, such as:

[TestMethod]

// Convert 10 gallons to quarts

public void GalToQts()

{

double result = mpnUOMConverter.ConvertUnits(10, mpnUnits.gal, mpnUnits.qt);

Assert.AreEqual(40, result);

}

Of course, this doesn’t work if trying to convert across domains, such as volume to mass.

[TestMethod]

[ExpectedException(typeof(InvalidOperationException))]

public void GalToLb()

{

double result = mpnUOMConverter.ConvertUnits(10, mpnUnits.gal, mpnUnits.lb);

}

But you can do these more complex conversions with an overload of the method that accepts a “using” factor:

[TestMethod]

// Convert 10 gallons into pounds, using 8 pound/gallon

public void GalToLbWithConversionFactor()

{

double result = mpnUOMConverter.ConvertUnits(10, mpnUnits.gal, mpnUnits.lb,

8.0, mpnUOMConverter.CreateCompositeUnit(mpnUnits.lb, mpnUnits.gal));

Assert.AreEqual(80, result);

}

[TestMethod]

// Convert 10 gallons into pounds, using .125 gallons/pound

public void GalToLbWithInvertedConversionFactor()

{

double result = mpnUOMConverter.ConvertUnits(10, mpnUnits.gal, mpnUnits.lb,

0.125, mpnUOMConverter.CreateCompositeUnit(mpnUnits.gal, mpnUnits.lb));

Assert.AreEqual(80, result);

}

[TestMethod]

// Convert 80 pounds into gallons using 8 pounds/gallon

public void LbToGalWithConversionFactor()

{

double result = mpnUOMConverter.ConvertUnits(80, mpnUnits.lb, mpnUnits.gal,

8.0, mpnUOMConverter.CreateCompositeUnit(mpnUnits.lb, mpnUnits.gal));

Assert.AreEqual(10, result);

}

Note that these cross-domain conversions can determine if the “using” factor needs to be inverted, so it doesn’t require the user or coder to determine whether to invert the conversion factor or not. This makes it very easy to have a product defined as “10-34-00 fertilizer, applied in gal/acre, purchased in tons, with a density of 10.4 gallons/pound”, and easily acquire purchase quantity from an application quantity.

The cross-domain conversions above used simple units for the source and target units. You can also use compound units, such as:

// Convert 160 dry ounces per acre to gal/hectare using 2.5 pounds/quart

public void OzPerAcreToGalPerAcreAcresUsingLbPerGal()

{

double result = mpnUOMConverter.ConvertUnits(160,

mpnUOMConverter.CreateCompositeUnit(mpnUnits.oz, mpnUnits.acre),

mpnUOMConverter.CreateCompositeUnit(mpnUnits.gal, mpnUnits.ha),

2.5, mpnUOMConverter.CreateCompositeUnit(mpnUnits.lb, mpnUnits.qt));

Assert.AreEqual(3.95, Math.Round(result, 2));

}

In this example, we changed domains (dry ounces to gallons) and systems (acres to hectare) using a conversion factor with neither numerator nor denominator present in the source or target unit.

You can even use a simple “using” factor, when resolving a conversion between a simple source and compound target unit:

[TestMethod]

// Calculate the total gallons needed to do 50 acres at 10 gallons/acre

public void GalPerAcreToGalUsingAcres()

{

double result = mpnUOMConverter.ConvertUnits(10,

mpnUOMConverter.CreateCompositeUnit(mpnUnits.gal, mpnUnits.acre),

mpnUnits.gal,

50, mpnUnits.acre);

Assert.AreEqual(500, result);

}

While enumerations are great for a developer using code completion technology, they must be stored in a database as integers, and that is not very friendly. It also prevents from reporting from a database directly w/o having access to a component to convert the enumeration back to text. And, of course, the enumerations define the simple units. A compound unit is a denominator enumeration shifted left 16 bits, or’d with the numerator unit. If you don’t know that that means, just trust that it generates a very ugly number for something like seeds/acre.

To make the system more flexible, we started supporting text definitions as an alternative to enumerations. Our rule is that the ‘/’ symbol is reserved to be a numerator/denominator separator, and that we should be able to split any text unit at the ‘/’ character and process the left and right halves into simple enumerations. So, “pound/acre” is treated as “pound” for a numerator and “acre” for a denominator, and the relative enumerated values are: mpnUnits.lb and mpnUnits.acre. We added overloads to all of the conversion methods so that a user can pass in any of these three values for “pound”:

* The enumeration: mpnUnits.lb
* The string “pound”
* The string “pounds”
* The string “lb”
* The string “lbs”
* The string “516” where 516 is the numeric representation of the enumeration mpnUnits.lb

With this change, an organization could choose to persist the unit as a string in the database and stay with enumeration values (the string version of the number) or human friendly text.

And, finally, we found that we were doing so much work with Deere, it would be easiest to make our UOM library support the Deere syntax. Deere’s definition of “pounds/acre” is “lb1acre-1”, where the numbers represent “to the nth power” and of course, a negative power is an inversion.

So, in addition to the above 6 representations for pound, we also support “lb1”.

We categorize the units that we support into domains and systems (and if you are a fan of bit twiddling, you’ll pick up on this in the definitions of the enumerations:

[Flags]

public enum mpnDomains

{

None = 0,

Length = 1,

Area = 2,

Volume = 4,

Mass = 8,

Time = 0x10, //'16

Count = 0x20, // '32

Commodity = 0x40, //64

Voltage = 0x80, //128

Current = 0x100, //256

Resistance = 0x200, //512

Conductivity = 0x400, //1024

Power = 0x800, //2048

Energy = 0x1000, //4096

Temperature = 0x2000, //8192

Concentration = 0x4000,

Pressure = 0x800000, //Need to be exotic on pressure because

//we are past the lower nibble.

Force = 0x01000000, //Same thing on Force... get into the outer nibble.

Mask = 0x700F,

All = 0x7FFF,

}

As you review the list, I believe you will find it very complete for “normal” stuff like mass, volume, area, and length. They are probably incomplete for some of the power and energy variants. Things like a Degree (angular) are treated like a count because we didn’t think of any built-in conversion that made sense outside of a projection and this is not a projection engine. And, then there are some units that appear as compound units when presented. For instance, meq/100g is visually presented as a compound unit, but it is represented as a simple enumeration within the “none” domain because we didn’t want to deal with related (unknown) conversions. The other significant one is “lb/acre (soil test)” as a “Concentration” unit instead of a compound unit. A soil test level is represented in this metric, and as a concentration, it is directly convertible to parts-per-million or percent. This is different than the compound unit of “lb/acre”, which is the numerator pound and denominator acre, and which cannot be converted to parts-per-million w/o defining the pounds in an acre. Convention is to use 2 million pounds in a 6” acre slice, hence the traditional doubling of a soil test value in ppm to get pounds/acre (soil test).

The actual units are:

[Flags]

public enum mpnUnits

{

//No units

none = 0x0,

unknown = 0x10,

meq100g = 0x30, //meq/100g -- Cation Exchange Capacity

nmtds100cc = 0x40, //Nematodes/100cc

mSm = 0x50, // milliSiemens/m

eggs100cc = 0x60, // eggs/100cc

//Length

be just the domain

cm = 0x11, //Base Unit

m = 0x21,

km = 0x31,

mm = 0x41,

inch = 0x101,

ft = 0x201,

mile = 0x301,

yd = 0x401,

usft = 0x501,

//Area

cm2 = 0x12, //Base Unit

m2 = 0x22,

km2 = 0x32,

ha = 0x42,

mm2 = 0x52,

in2 = 0x102,

ft2 = 0x202,

mile2 = 0x302,

acre = 0x402,

yd2 = 0x502,

usft2 = 0x602,

kft2 = 0x702,

//Volume

ml = 0x13,

l = 0x23,

m3 = 0x33,

cm3 = 0x43,

mm3 = 0x53,

floz = 0x103,

pt = 0x203,

qt = 0x303,

gal = 0x403,

bu = 0x503,

in3 = 0x803,

ft3 = 0x903,

acreinch = 0xA03,

kgal = 0xB03,

//Mass

g = 0x14,

kg = 0x24,

tonne = 0x34,

mg = 0x44,

oz = 0x104,

lb = 0x204,

cwt = 0x304,

ton = 0x404,

fiftylbbag = 0x504,

sixtylbbag = 0x604,

longton = 0x704,

troylb = 0x804,

troyoz = 0x904,

//Time

sec = 0x115,

min = 0x125,

hour = 0x135,

day = 0x145,

week = 0x155,

month = 0x165,

year = 0x175,

//Count

unit = 0x16,

bag = 0x26,

seed = 0x36,

can = 0x46,

box = 0x56,

bottle = 0x66,

cse = 0x76,

revolution = 0x86,

index = 0x96,

meq = 0xA6,

bale = 0xB6,

bigbale = 0xC6,

load = 0xD6,

degree = 0xE6,

//Commodity

//Voltage

volt = 0x28,

//Elecitrical Current

Amp = 0x19,

//Resistance

Ohm = 0x1A,

//Conductivity

mS = 0x1B, //milliSiemen

//Power

watt = 0x1C,

kw = 0x2C, //kilowatt

hp = 0x3C,

ftlbsec = 0x4C,

//Energy

kwh = 0x1D, //kilowatt hour

hph = 0x2D, //horsepower hour

//Temperature

celsius = 0x1E,

fahrenheit = 0x2E,

//Concentration

percent = 0x1F,

ppm = 0x2F,

lbsacre = 0x3F,

//Pressure

pascal = 0x1010,

psi = 0x1100,

inh2o = 0x1020,

//Force

newton = 0x2010,

poundforce = 0x2100,

}