Библиотека Разработчика PHYSICS C# Версия 5.0

Руководство

Copyright © Sergey L. Gladkiy
Email: lrndlrnd@mail.ru
URL: www.Sergey-L-Gladkiy.narod.ru

Содержание

Введение

PHYSICS C# – это библиотека для разработчиков. Библиотека PHYSICS содержит специальные классы и позволяет использовать различные физические понятия (например, физические величины, единицы измерения и др.) в программах .NET.

ПРЕИМУЩЕСТВА библиотеки PHYSICS:

1. 100% код C#
2. Строго структурированная иерархия классов (наиболее точно приближенна к современным представлениям физики).
3. Универсальные алгоритмы для работы с физическими понятиями (без необходимости изменения первичного исходного кода).
4. Множество предопределенных классов физических объектов (физические величины, единицы измерения).
5. Простота добавления новых классов физических объектов (физических величин, единиц измерения).
6. Простота локализации любых физических объектов.
7. Работа с физическими значениями – скалярными, векторными, тензорными.
8. Поддержка логарифмических единиц измерения и децибел.

Зависимости

Библиотека PHYSICS C# зависит от:

1. Библиотеки NRTE (NET Run-Time Environment).

Расширения

Существуют следующие расширения библиотеки PHYSICS C#:

1. Measurement (реализует множество производных единиц измерения и физических величин).
2. Mathphysics (реализует скалярные, векторные и тензорные физические значения).

1. Физические величины, единицы измерения и значения

Физические величины

Физическая величина является свойством физического вещества, которое может быть измерено (http://en.wikipedia.org/wiki/Physical_quantity). Примеры физических величин: масса, время, электрический ток и т.д. Следует различать понятия физической величины и значения физической величины. Физическая величина это абстрактное понятие, а значение физической величины является конкретным значением некоторого свойства. Например, **5 г** является значением физической величины **масса** (о физических величинах см. ниже).

Существуют фундаментальные (базовые) физические величины, которые называются размерностями: **длина**, **время**, **масса**, **температура**, **электрический ток**, **количество вещества**, **сила света**. Они являются фундаментальными, в том смысле, что все другие величины могут быть выражены через них. Например, физическая величина скорость может быть выражена через длину и время: **скорость = длина/время**. Таким образом, любая физическая величина имеет атрибут, называемый размерностью. Величина скорости имеет размерность **длина/время**, ускорение имеет размерность **длина/время**^{2}, площадь имеет размерность **длина**^{2} и т.д.

Таким образом, любую физическую величину можно охарактеризовать двумя свойствами: имя (или символ) и размерность. Существуют фундаментальные величины с теми же размерностями и производные величины (размерности которых выражены через фундаментальные).

ПРИМЕЧАНИЕ. Можно ввести три дополнительные размерности (фундаментальные физические величины): **информация**, **плоский угол**, **телесный угол**. В действительности эти величины являются безразмерными, но могут рассматриваться как дополнительные размерности, чтобы отличать их от действительно безразмерных данных (чисел).

Иерархия классов физических величин

В соответствии со сказанным ранее, иерархия классов физических величин в библиотеке PHYSICS соответствует понятиям современной физики. Каждая физическая величина представлена классом. Базовым абстрактным классом для всех физических величин является **PhysicalQuantity**. Он имеет три основных свойства: **Name**, **Symbol** и **Dimension**. **FundamentalQuantity** и **DerivedQuantity** являются прямыми наследниками этого базового класса. Десять фундаметальных величин (**Length**, **Time**, **Mass**, **Temperature**, **ElectricCurrent**, **AmountOfSubstance**, **LuminousIntensity**, **Information**, **PlaneAngle**, **SolidAngle**) наследуются от первого класса, а все другие от последнего. Схема иерархии классов показана на рисунке 1.1.

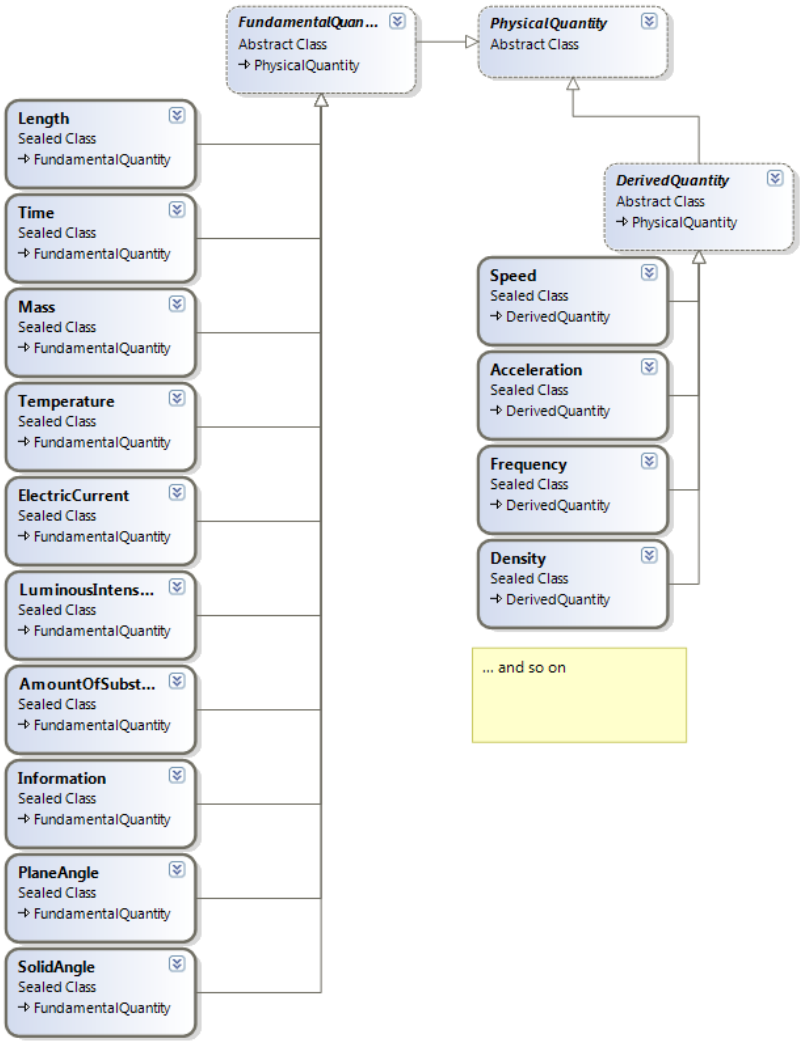

Рисунок 1.1. Схема иерархии классов физических величин.

Введение новой физической величины

Библиотека PHYSICS предоставляет простой механизм введения новой физической величины. Т.к. каждая величина представлена классом – необходимо унаследовать новый класс от базового класса **DerivedQuantity**. Например, чтобы ввести физическую величину ‘сила’, необходимо создать следующий класс

```
/// <summary>
/// Force
/// </summary>
public sealed class Force : DerivedQuantity
{
    protected internal override string GetName()
    {
        return "Force";
    }

    protected internal override string GetSymbol()
    {
        return "F";
    }

    protected internal override Dimension GetDimension()
    {
        return (Dimension.Mass * Dimension.Length) / (Dimension.Time ^ 2);
    }
}
```

Этот код демонстрирует ввод новой физической величины с именем 'Force' (символ 'F') и размерностью **mass·length/time²** (размерность физической величины ‘сила’)[1].

Этого достаточно для введения новой физической величины. Библиотека PHYSICS использует механизмы рефлексии .NET для идентификации классов физических объектов. Новый класс будет автоматически найден и зарегистрирован. Все алгоритмы работы с физическими величинами не зависят от типа величины. Программа сможет найти величину, отобразить ее (см. ниже о локализации), найти единицы измерения, совместимые с данной величиной (и даже автоматически создавать совместимые единицы измерения) и т.д.

Единицы измерения

Единица измерения - это определенное значение физической величины (http://en.wikipedia.org/wiki/Unit_of_measurement). Например, **метр** является предопределенным значением физической величины **длина**. Таким образом, как и физическая величина, каждая единица измерения имеет такой атрибут как **размерность**. Физическая величина может измеряться в некоторых единицах **тогда и только тогда**, когда они имеют **одинаковую** размерность. Таким образом, связь между физическими величинами и единицами измерения является связью типа "многие ко многим". То есть, одна физическая величина может измеряться разными единицами, и одна единица измерения может быть мерой разных физических величин. Например, масса может быть измерена граммами, фунтами, килограммами и др. С другой стороны, как гидростатическое давление, так и механическое напряжение можно измерять одной и той же единицей, например - Паскаль.

В соответствии со сказанным ранее, есть базовые единицы измерения для измерения фундаментальных физических величин. Для величины длина - метр, дюйм и так далее, для массы - грамм, килограмм и так далее. Все единицы измерения для производных величин могут быть получены путем умножения и деления базовых единиц, умножения их на некоторые константы и прибавления к ним постоянных значений (с помощью общих правил алгебры). Для примера, единицу измерения площади 'Are' можно получить умножением **10 m** на **10 m**, таким образом, **1 a = 100 m²**.

Одной из главных целей использования единиц измерения в программе является преобразование значений физических величин из одних единиц измерения в другие. Базовые величины преобразуются в соответствии с их определениями. Например, для преобразования температуры T из градусов Цельсия в градусы Фаренгейта используется следующее уравнение **t °C = 5/9 (t °F - 32)**. Все базовые единицы измерения являются "линейными", потому что они имеют единственную размерность. Таким образом, все они преобразуются по линейному закону. Закон преобразования производной единицы измерения определяется тем, как она получена из базовых единиц измерения. Например, **1 m² = 1 m · 1 m = 100 cm · 100 cm = 10000 cm²**.

Несмотря на то, что данный алгоритм преобразования единиц измерения понятен человеку, достаточно сложно написать программный код данного алгоритма. Главная трудность заключается в создании «правильной» иерархии классов, которая обеспечивает, с одной стороны, универсальный алгоритм для всех единиц измерения, а с другой – простой способ введения новых единиц измерения.

Библиотека PHYSICS имеет **уникальный универсальный алгоритм** преобразования единиц измерения. Он преобразует значение **любой единицы измерения в любую другую единицу измерения** (если они имеют одинаковую размерность). Единственное, что необходимо, это создать новый класс единицы измерения. Алгоритм будет конвертировать значения данной единицы измерения в любое другое, совместимое с ней.

---

[1] При использовании перегруженного оператора ‘^’ в коде вычисления размерности физической величины следует принимать во внимание, что оператор ‘^’ в языке C# имеет более низкий приоритет выполнения, чем операторы ‘*’ и ‘/’. Поэтому, данный оператор всегда следует брать в скобки, чтобы порядок операций соответствовал ‘математическому’.

Иерархия классов единиц измерения

Иерархия классов единиц измерения (не полностью) показана на рисунке 1.2. Она является достаточно сложной. Данная иерархия была построена с целью представления понятия «единицы измерения» максимально приближенного к современному понятию физики, а так же обеспечить универсальный алгоритм преобразования единиц измерения и наиболее простой способ введения новых единиц измерения.

Базовым абстрактным классом для всех единиц измерения является класс **Unit**. Он предоставляет общие свойства единиц измерения - **Name**, **Symbol**, **Dimension**, и общие алгоритмы – преобразования значения из одной единицы измерения в другую, проверку единиц измерения на совместимость и др. Так же, класс имеет перегруженные операторы: ==, !=, *, /, ^ (оператор степени), оператор неявного преобразования строк в единицы измерения.

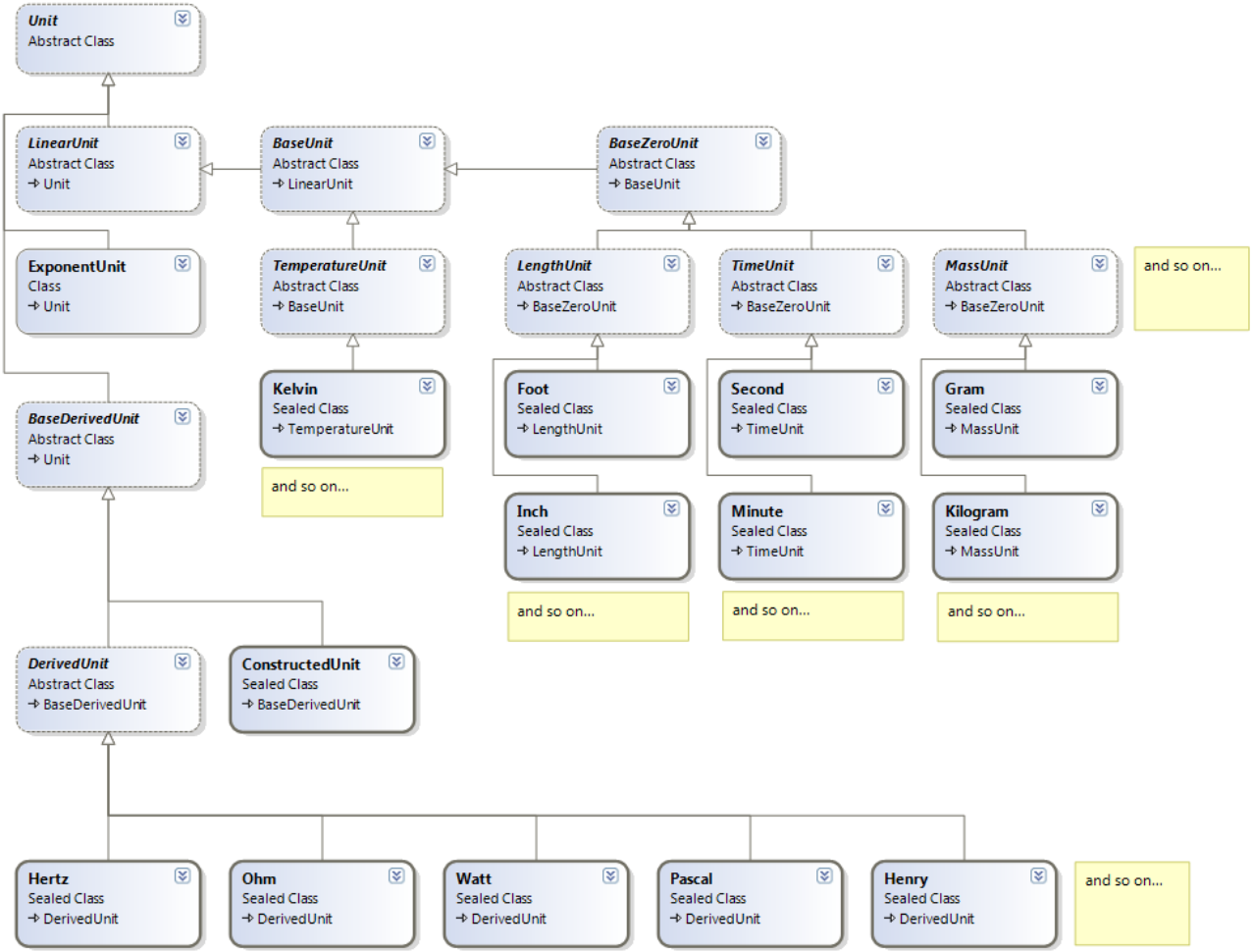

Рисунок 1.2. Диаграмма иерархии классов единиц измерения.

Класс **LinearUnit** является прямым наследником базового абстрактного класса **Unit** и реализует абстрактные методы для единиц измерения, которые преобразуются по линейному закону. Как было сказано выше, все единицы измерения для фундаментальных физических величин (размерностей) линейны, поэтому класс **BaseUnit** реализует абстрактные механизмы таких единиц измерения. Базовые единицы разделены на одиннадцать классов, соответствующих фундаментальным величинам (десять размерностей + безразмерная). Все единицы измерения, соответствующие фундаментальным величинам должны быть унаследованы от данных одиннадцати единиц. Библиотека содержит множество предопределенных базовых величин: **Foot**, **Gram**, **Second** и др.

Все остальные единицы измерения строятся из базовых. Самым простым способом получить новую единицу измерения является умножение ее на некоторое число и/или возведение ее в степень. Например, из единицы измерения метр **m** может быть построена новая единица 10 m^2. Данную концепцию реализует класс **ExponentUnit**. Для реализации стандартных множителей единиц измерения (http://en.wikipedia.org/wiki/SI_prefix) предназначен специальный абстрактный класс **UnitPrefix** (рисунок 1.3). В библиотеке PHYSICS реализованы все стандартные множители единиц измерения.

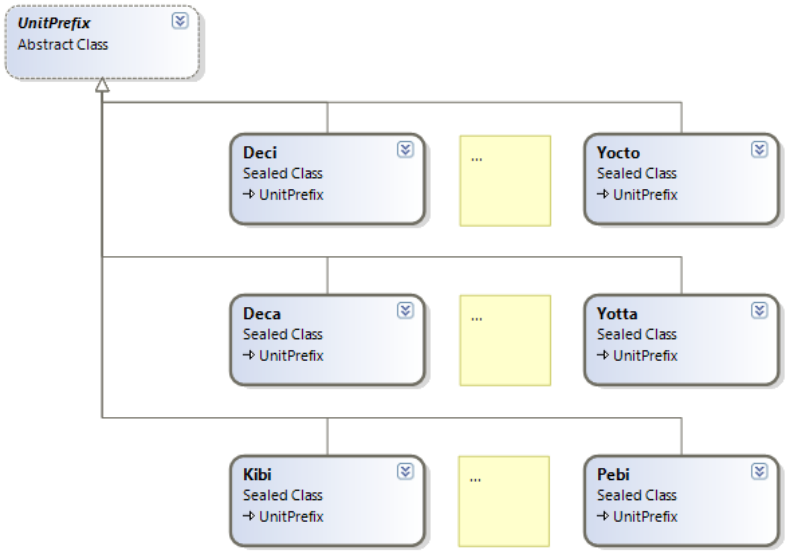

Рисунок 1.3. Диаграмма классов префиксов единиц измерения.

И, наконец, самый общий способ получения новой единицы измерения из базовых единиц состоит в умножении нескольких единиц измерения, возведенных в некоторую степень. Например, **kg·m²·A⁻¹·s⁻²**. Эта концепция реализована в классе **BaseDerivedUnit**. Все производные единицы измерения можно разделить на две категории. Первые – единицы измерения, имеющие свои собственные имена. Например - Newton, Pascal, Henry и др. Вторые – единицы измерения, не имеющие собственных имен, например **km·s⁻²**. Данные концепции реализованы в классах **DerivedUnit** и **ConstructedUnit** соответственно.

В библиотеке PHYSICS существует множество предопределенных единиц измерения, новые единицы измерения могут быть легко добавлены в библиотеку без необходимости изменения алгоритма их использования.

Ввод новых единиц измерения

Иерархия классов единиц измерения является достаточно сложной, но для ввода новых единиц измерения в программу и их использования не обязательно знать все особенности ее структуры. Библиотека PHYSICS предоставляет простые механизмы по введению и использованию новых единиц измерения.

Есть два различных способа использования новых единиц измерения в программе. Первый заключается в создании новых классов единиц измерения. Этот способ должен быть использован для всех единиц, которые имеют собственные названия (Pascal, Newton и т.д.). Для введения единиц измерения, соответствующих базовым физическим величинам, класс должен наследоваться от одной из базовых единиц измерения. В качестве примера приведен код единицы измерения 'ярд'.

```
/// <summary>
/// Yard = 0.9144 m
/// </summary>
public sealed class Yard : LengthUnit
{
    protected internal override double GetFactor()
    {
        return 0.9144;
    }

    protected internal override string GetName()
    {
        return "Yard";
    }

    protected internal override string GetPlural()
    {
        return "Yards";
    }

    protected internal override string GetSymbol()
    {
        return "yd";
    }

}
```

Единица измерения **Yard** наследуется от **LengthUnit**, потому что она представляет собой меру физической величины **длина**. В классе новой единицы измерения 'ярд' переопределены методы, возвращающие имя и символ единицы измерения. Также переопределен метод **GetFactor**, который возвращает коэффициент преобразования данной единицы в базовую единицу измерения. Это

единственная информация, необходимая для алгоритма конвертации. Все коэффициенты должны соответствовать преобразованию данной единицам измерения в основную единицу системы СИ (http://en.wikipedia.org/wiki/SI).

Все сложные единицы измерения, должны быть непосредственно унаследованы от класса **DerivedUnit**. В качестве примера приведен код единицы измерения "ватт".

```
/// <summary>
/// Watt ( kg m^2/s^3 )
/// </summary>
public sealed class Watt : DerivedUnit
{
    protected internal override string GetName()
    {
        return "Watt";
    }

    protected internal override string GetPlural()
    {
        return "Watts";
    }

    protected internal override string GetSymbol()
    {
        return "W";
    }

    protected internal override void RecreateUnits()
    {
        fUnits = new ExponentUnit[3];
        fUnits[0] = new ExponentUnit(new Kilogram(), 1);
        fUnits[1] = new ExponentUnit(new Metre(), 2);
        fUnits[2] = new ExponentUnit(new Second(), -3);
    }
}
```

Класс переопределяет методы, возвращающие имя и символ, а также переопределен метод **RecreateUnits**, который обеспечивает основную информацию для алгоритма конвертации и других алгоритмов. В данном примере единица измерения 'ватт' построена из трех экспоненциальных единиц измерения **Kilogram**^{1}·**Metre**^{2}·**Second**$^{-3}$.

Вторым способом использования новых единиц измерения в программе является создание экземпляров объектов динамически во время выполнения. Самый простой способ – использовать перегруженные операторы для конструирования нового класса. Следующий код демонстрирует создание единицы измерения во время выполнения.

```
Unit u1 = new Kilogram();
Unit u2 = new Metre();
Unit u3 = new Second();
Unit x = u1 * u2 * (u3 ^ -2);
```

Созданная единица измерения x = **kg·m/s**^{2}.[2]

Преобразование единиц измерения

Библиотека PHYSICS позволяет легко выполнять преобразование единиц измерения. Базовый класс **Unit** имеет статический метод **Convert** для преобразования значения из любой единицы измерения в любую совместимую с ней. Следующий код демонстрирует пример преобразования

```
Unit u1 = new Pascal();
Unit u2 = new Atmosphere();
double x1 = 100;
double x2 = Unit.Convert(u1, u2, x1);
```

В этом примере значение типа 'double' преобразуется из единиц измерения **Pascal** в единицы измерения **Atmosphere**. Для выполнения преобразования, единицы измерения должны быть совместимы. Единицы измерения являются совместимыми, если они имеют одинаковые размерности. Для проверки совместимости единиц измерения можно использовать статический метод **Convertible** класса **Unit**.

Быстрая конвертация единиц измерения

Преобразование единиц измерения, описанное выше, является универсальным и может быть применено к любым (конвертируемым) единицам измерения. Но данный алгоритм может быть медленным при выполнении преобразований огромных массивов значений. Чем сложнее единицы измерения, тем медленнее преобразование.

В то же время, все единицы измерения можно разделить на две категории[3]: "с нулевым сдвигом" и "с ненулевым сдвигом". Единицами "с нулевым сдвигом" являются такие, у шкал которых совмещены нули. Например, единицы измерения массы относятся к единицам "с нулевым сдвигом", поскольку нулевая масса всегда остается нулевой, в независимости от того, в каких единицах она

[2] При использовании перегруженного оператора '^' в коде создания новой единицы измерения следует принимать во внимание, что оператор '^' в языке C# имеет более низкий приоритет выполнения, чем операторы '*' и '/'. Поэтому, данный оператор всегда следует брать в скобки, чтобы порядок операций соответствовал 'математическому'.

[3] Другие случаи рассмотрены в разделе 'Специальные единицы измерения'.

измерена. И наоборот, единицы измерения температуры являются единицами "с ненулевым сдвигом", потому что температура **0°C не равна 0°K**.

Преобразование любых единиц измерения "с нулевым сдвигом" всегда линейно и может быть выполнено с помощью одного множителя **f**. Формула преобразования в этом случае имеет вид $\mathbf{x\ unit_1 = f \cdot x\ unit_2}$, где **x** - это значение, **unit1** и **unit2** – единицами измерения, для которых выполняется конвертация, **f –** коэффициент преобразования. Класс **Unit** содержит метод **IsZeroBased**, проверяющий, является ли данная единица измерения единицей "с нулевым сдвигом".

Преобразование единицы измерения "с ненулевым сдвигом" является более сложным и может быть не линейно. Однако, когда необходимо выполнить преобразование интервалов, все единицы измерения преобразуются по линейному закону, путем умножения значения на некоторый "масштабный коэффициент". Например, температурный интервал **10°C** может быть преобразован в интервал температур, измеренный в градусах Фаренгейта, путем его умножения на **9/5**.

В соответствии со сказанным выше, для единиц измерения "с нулевым сдвигом" и "интервального" преобразования может быть использован один множитель для преобразования значения одной единицы измерения в другую. Класс **Unit** содержит статический метод **ScaleFactor** для вычисления данного множителя. Ниже приведен пример кода, демонстрирующий данный способ преобразования для массива значений.

Пусть есть массив значений давления, которые измерены в "миллиметрах ртутного столба".

```
double[] pressures;
```

И все значения должны быть преобразованы к единицам измерения Pascal. Код преобразования может быть следующим:

```
Unit x1 = new MillimetreOfMercury();
Unit x2 = new Pascal();
double z = Unit.ScaleFactor(x1, x2);
for (int i = 0; i < pressures.Length; i++)
{
    pressures[i] = z * pressures[i];
}
```

Преобразование строк в единицы измерения

Библиотека PHYSICS обеспечивает механизмы для преобразования строковых данных в единицы измерения. Данные преобразования могут быть использованы, если единицы измерения некоторых данных должен вводить (или выбирать) пользователь программы. Методы преобразования строк в единицы измерения предоставляются классом **UnitConverter**. Следующий код демонстрирует преобразование строки в единицу измерения.

```
UnitConverter uconverter = new UnitConverter();
string ustr = "N mm^2/ns";
string error;
Unit u = uconverter.ConvertString(ustr, out error);
```

В данном примере, новая единица измерения x получена путем преобразования строки. В результате, полученная единица измерения имеет значение **Newton·millimeter²/nanosecond**.

Правила преобразования строк в единицы измерения:

- Строка может содержать символы единиц измерения, префиксы, пробелы, знак оператора деления и знак оператора степени.
- Только один знак деления может быть в строке.
- Префиксы не отделяются пробелами от единиц измерения, которым они принадлежат.
- Экспоненциальные единицы измерения разделяются пробелами (то есть пробелы используются в качестве оператора умножения).

Если в строке имеется ошибка, ее описание будет возвращено в параметре функции error.

Специальные единицы (логарифмические и децибелы)

Иерархия классов единиц измерения, представленная выше, не является полной. Диаграмма была сокращена для упрощения. Существует еще одна ветвь наследования единиц измерения (рис. 1.4).

Главный узел этой ветви наследования – класс **SpecialUnit**. Данная ветвь иерархии была введена для специальных целей. Класс **SpecialUnit** используется наряду со всеми другими классами единиц измерения в алгоритмах ядра библиотеки, но использование предполагает особые правила. Примером специальных единиц измерения является децибел – это логарифмическая единица измерения со специальными правилами преобразования и операциями (https://en.wikipedia.org/wiki/Decibel). Децибел является логарифмической единицей измерения по основанию 10, поэтому унаследован от класса **Decimal**. По определению, логарифмические единицы безразмерны, поскольку выражают относительные значения. Однако, можно ввести размерные децибелы, если зафиксировать опорное значение для отношения (https://en.wikipedia.org/wiki/Decibel#Suffixes_and_reference_values).

Особенностью логарифмических единиц является то, что они НЕ МАСШТАБИРУЕМЫ. Это значит, что 'интервальное' преобразование значения к другим единицам не может быть выполнено с помощью масштабного множителя (в общем случае). Преобразование может быть выполнено только общим алгоритмом (быстрое преобразование не возможно). Для определения, может ли единица измерения быть преобразована масштабированием, используется метод **IsScalable** класса Unit. Так же, метод **MakeConversion** может быть использован для получения делегата правильного преобразования с учетом свойств единиц измерения.

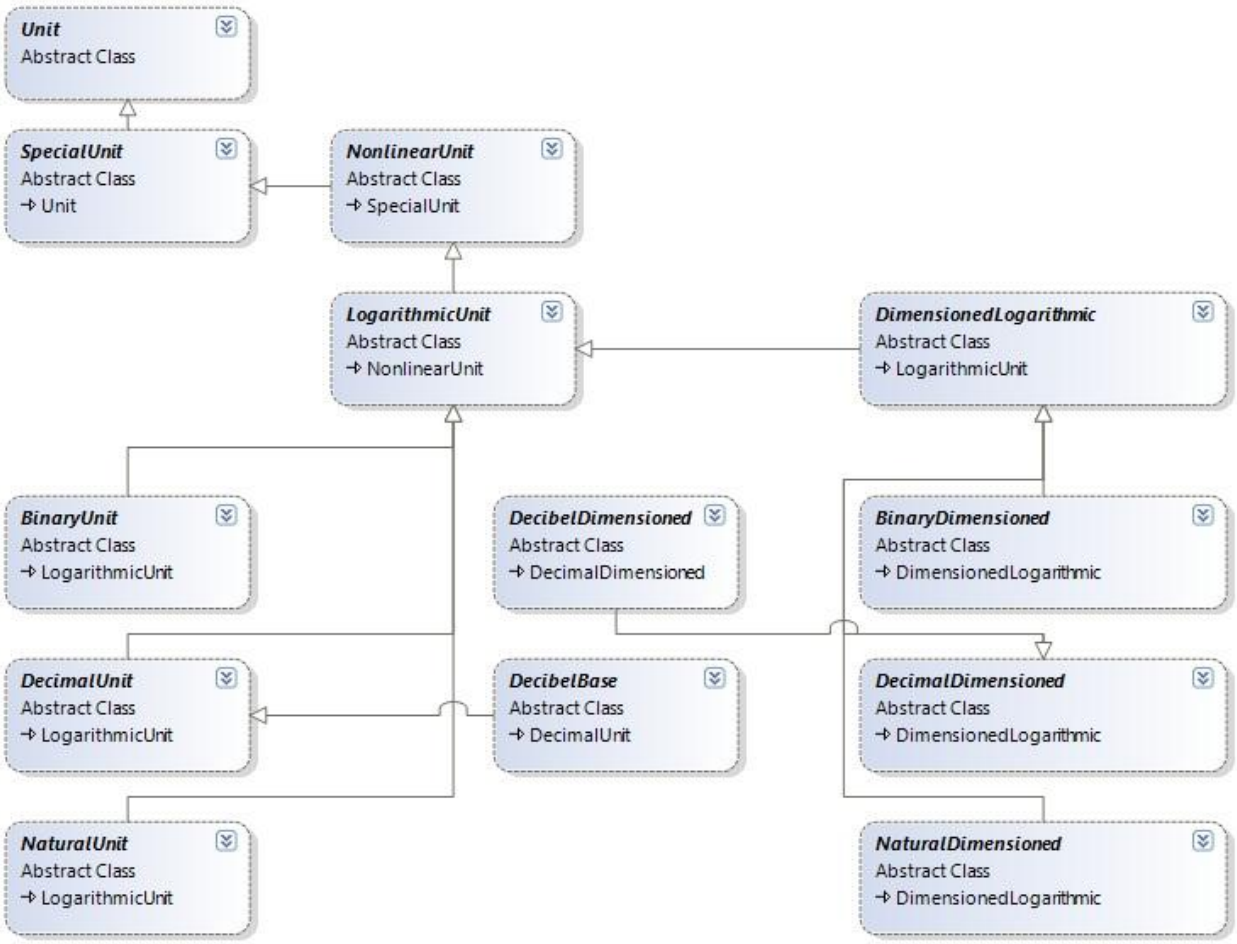

Рисунок 1.4. Диаграмма наследования специальных единиц измерения.

Свод особенностей использования логарифмических единиц и децибелов:

- Логарифмические единицы являются немасштабируемыми с другими единицами.
- Децибелы могут быть как безразмерными, так и размерными.
- Логарифмические единицы и децибелы нельзя умножать с другими логарифмическими единицами.
- Логарифмические единицы и децибелы нельзя возводить в степень (кроме 1).
- При выполнении операций с физическими значениями (см. ниже) логарифмические значения имеют особенности: логарифмические значения нельзя умножать; логарифмические значения нельзя возводить в степень; логарифмические значения можно суммировать, если они имеют одну размерность или одно из них безразмерно.

Физические значения (значения физических величин)[4]

Физическое значение это значение некоторой физической величины, которое представлено числовым значением и единицей измерения. Например, **5 gram** является значением физической величины **масса**. С физическими значениями могут быть выполнены алгебраические операции. Например, можно сложить два физических значения (если их единицы измерения совместимы друг с другом) **2 kg + 5 g = 2005 g** или умножить (любые) два значения **2 m · 3 s⁻² = 6 m/s²**.

Базовым абстрактным классом для всех физических значений является **PhysicalValue**. Данный класс предоставляет базовые данные (единицы измерения) и функциональность (преобразование из строки и наоборот). Производный класс **ScalarValue** переопределяет функциональность базового класса и расширяет ее, обеспечивая возможности работы со скалярными физическими величинами. Следующий код демонстрирует использование скалярных значений

```
string s1 = "9.8 m/s^2";
string s2 = "70.5 kg";
ScalarValue v1 = ScalarValue.Parse(s1);
ScalarValue v2 = ScalarValue.Parse(s2);
ScalarValue f = v1 * v2;
string s = f.ToString();
```

В примере создаются два скалярных значения v1 и v2 (с помощью преобразования строковых значений). Первое значение соответствует ускорению, а второе соответствует массе. При умножении этих двух значений, получится новое значение f, которое соответствует силе. В результате строка s будет иметь значение “690.9 m kg/s^2”.

[4] Классы физических значений реализованы в сборке **Mathphysics**.

Аналогично могут быть введены физические значения других типов – векторные, тензорные, матричные и т.д. В библиотеке полностью реализованы классы **VectorValue** и **TensorValue** (для 3D векторов и тензоров соответственно) с перегрузкой операторов и функцией **Parse**.

Далее приведен пример кода использования векторных и тензорных значений:

```
VectorValue s1 = VectorValue.Parse("(0 1 0) m");
VectorValue s2 = VectorValue.Parse("(203e-1 0 0) ft");
VectorValue F = VectorValue.Parse("(2 1 -1) N");
ScalarValue W = VectorValue.Dot(F, s1+s2);
…
TensorValue x1 = TensorValue.Parse("(1 2 -1 0 1 -2 1 1 0) mm g");
TensorValue x2 = TensorValue.Parse("(0 -1 1 1 1 -2 1 1 0) s");
TensorValue s = x1/x2;
```

Вычисленные значения результирующих величин:

```
W = 13.37488 N m
s = (-1 0 1 -1 0.5 -0.5 0 0 1) mm g/s
```

Правила выполнения операций с физическими значениями:

- Два физических значения можно складывать, если их значения математически могут суммироваться, а единицы измерения совместимы (имеют одну размерность).
- При сложении физических значений они приводятся к единицам измерения первого.
- Если выполняется преобразование, оно выполняется как ‘интервальное’ (поскольку, например, складывая 1 метр и 2 сантиметра подразумевается сложение интервалов), если оно возможно.
- При сложении логарифмических значений, они преобразуются по общему алгоритму.
- При умножении значений с единицами одинаковых размерностей, результат выражается через квадрат первой единицы измерения (исключение – безразмерные единицы в первой степени).
- При делении значений одинаковой размерности результат выражен в безразмерном виде (в долях единицы) (исключение – безразмерные выражены в единицах первого значения).

Локализация

Библиотека PHYSICS предоставляет простой механизм локализации всех отображаемых значений. Для локализации единиц измерения и физических величин используются классы **LocalizedResourceUnitConverter** и **LocalizedResourceQuantitiesManager** соответственно. Для локализации преобразований физических значений, глобальному статическому свойству **PhysicalValue.UnitConverter** должно быть присвоено значение экземпляра класса **LocalizedResourceUnitConverter**.

Классы локализации использует стандартный механизм ресурсов .NET, для локализации значений для определенный культур и языков. Имена ресурсов идентифицируются именами классов и именем свойства. Например, для локализации свойства Symbol единицы измерения Pascal, должен быть создан ресурс с именем "Pascal_Symbol". Этот ресурс должен быть предоставлен экземпляру класса **LocalizedResourceUnitConverter** через параметр в его конструкторе.

Библиотек PHYSICS содержит локализацию всех классов для русского языка. Данная локализация может быть использована в качестве примера для локализации других языков.

Замечание: если для данной культуры (языка) некоторый ресурс не определен, используется значения по умолчанию (то есть значение, определенное в коде). Таким образом, максимум два значения могут быть использованы для любого символа одновременно - значение по умолчанию и локализованное.

