UnitParser

DOI 10.5281/zenodo.803378

Last release -- Master source code

Main page (versión en español) -- Code analysis -- NuGet package -- Video

Introduction

The main class is called *UnitP* (*FlexibleParser* namespace). It can be instantiated in many different ways.

```
//1 N.
UnitP unitP = new UnitP("1 N");

//1 N.
unitP = new UnitP(1m, UnitSymbols.Newton);

//1 N.
unitP = new UnitP(1m, "nEwTon");

//1 N.
unitP = new UnitP(1m, Units.Newton);
```

UnitP can be seen as an abstract concept including many specific types (full list). Same-type variables can be added/subtracted. Different-type variables can be multiplied/divided, but only in case of generating a valid-type output.

```
//2 N.
unitP = new UnitP("1 N") + new UnitP(1m, Units.Newton);

//1 J.
unitP = new UnitP("1 N") * new UnitP("1 m");

//Error not triggering an exception.
//The output unit N*m^2 doesn't match any supported type.
unitP = new UnitP("1 N") * new UnitP("1 m") * new UnitP("1 m");
```

Main Variable Information

UnitP variables are defined according to various *readonly* fields populated at instantiation.

Unit - Corresponding Units member.
UnitType - Corresponding UnitTypes member.
UnitSystem - Corresponding UnitSystems member.
UnitParts - Defining parts of the given unit.

UnitPrefix - Supported prefix affecting all the unit parts.
BaseTenExponent - Base-ten exponent used when dealing with too small/big values.
Error - Variable storing all the error- and exception-related information.

General Rules

All the functionalities are based upon the following ideas:

- In case of incompatibility, the first element is always preferred.
- By default, the formally-correct alternative is preferred. Some required modifications might be performed.
- By default, all the errors are managed internally.

```
//1.3048 m.
unitP = new UnitP("1 m") + new UnitP("1 ft");

//Error not triggering an exception.
//The parser expects "km" or a full-name-based version like "KiLom".
unitP = new UnitP("1 Km");

//999999.999999900000 * 10^19 YSt.
unitP = 999999999999999999999999999999999999.9 * new UnitP("9999999999999 St");
```

Unit String Parsing Format

The unit string parsing part is quite flexible, but there are some basic rules.

- String multi-part units are expected to be exclusively formed by units, multiplication/division signs and *integer* exponents.
- Only one division sign is expected. The parser understands that all what lies before/after it is the numerator/denominator.

```
//1 W.
unitP = new UnitP("1 J*J/s*J2*J-1*s*s-1");

//Error not triggering an exception.
//The parser understands "J*J/(s*J2*s*J*s)", what doesn't represent a supported type.
unitP = new UnitP("1 J*J/(s*J2*s)*J*s");
```

Numeric Support

Formally, two numeric types are supported: *decimal*, almost everywhere; and *double*, only in multiplication/division with *UnitP* variables. Practically, *UnitP* variables implement a mixed system delivering *decimal* precision and beyond-*double*-range support.

```
//7.891011 ft.
unitP = new UnitP("1 ft") * 7.891011m;

//1.213141516 Gs.
unitP = new UnitP("1 s") * 1213141516.0;

//0.0003094346047382564187534467*10^-752 ym.
unitP = 0.00000000000000000000000000000000000000000000000000001 * new
UnitP(0.000000000000000000001m, "ym2") /
new UnitP("999999999999999999999 Ym") / double.MaxValue / double.MaxValue;
```

Further Code Samples

The test application includes a relevant number of descriptive code samples.

Authorship & Copyright