Users's Manual for Brent's Root Ada Package

Brent Seidel
Phoenix, AZ

July 23, 2024

Note that this is a draft version and not the final version for publication.

Contents

1 Introduction **1**

2 How to Obtain **2**

2.1 Dependencies 2

2.1.1 Ada Libraries 2

2.1.2 Other Libraries 2

3 Usage Instructions **3**

4 API Description **4**

4.1 `BBS` 4

4.2 `BBS.Units` 5

Chapter 1

Introduction

This project is the root project for most of my other Ada projects. It provides a namespace, `BBS`, that the other projects live under. It also provides some common types, units, and conversions that the other projects can use.

Chapter 2

How to Obtain

This collections is currently available on GitHub at `https://github.com/BrentSeidel/BBS-Ada`.

2.1 Dependencies

2.1.1 Ada Libraries

The following Ada libraries are used:

- `Ada.Unchecked_Conversion`

2.1.2 Other Libraries

There are no other dependencies.

Chapter 3

Usage Instructions

This is a library of routines intended to be used by some program. To use these in your program, edit your `*.gpr` file to include a line to `with` the path to `bbs.gpr`. Then in your Ada code `with` in the package(s) you need and use the routines.

Chapter 4

API Description

4.1 BBS

This package defines the following types:

- `bit` as `range 0 .. 1 with Size => 1`
- `int8` as `range -128 .. 127 with size => 8`
- `uint8` as `mod 2**8 with size => 8`
- `int16` as `-(2**15) .. 2**15 - 1 with size => 16`
- `uint16` as `mod 2**16 with Size => 16`
- `int32` as `-(2**31) .. 2**31 - 1 with Size => 32`
- `uint32` as `mod 2**32 with Size => 32`
- `int64` as `range -(2**63) .. 2**63 - 1 with Size => 64`
- `uint64` as `mod 2**64 with Size => 64`

And the following conversion functions:

```
function uint8_to_int8 is
     new Ada.Unchecked_Conversion(source => uint8, target => int8);
function int8_to_uint8 is
     new Ada.Unchecked_Conversion(source => int8, target => uint8);
function uint16_to_int16 is
     new Ada.Unchecked_Conversion(source => uint16, target => int16);
function int16_to_uint16 is
     new Ada.Unchecked_Conversion(source => int16, target => uint16);
function uint32_to_int is
      new Ada.Unchecked_Conversion(source => uint32, target => Integer);
function int_to_uint32 is
```

```
      new Ada.Unchecked_Conversion(source => Integer, target => uint32);
function uint64_to_int64 is
     new Ada.Unchecked_Conversion(source => uint64, target => int64);
function int64_to_uint64 is
     new Ada.Unchecked_Conversion(source => int64, target => uint64);
```

4.2 BBS.Units

This package defines a number of physical unites along with some conversions and operations. The units fall into the following categories:

Type	Prefix	Base Units
Length	len	meters
Area	area	meters^2
Volume	vol	liters
Mass	mass	kilograms
Force	force	Newtons
Temperature	temp	Celsius
Pressure	press	Pascal
Velocity	vel	m/s
Acceleration	acce	m/(s^2)
Angular	ang	radians
Rotation rate	rot	radians/second
Magnetic	mag	Gauss
Electromotive force	emf	Volt
Electrical current	curr	Amper
Electrical resistance	res	Ohms
Frequency	freq	Hertz
Time	time	Seconds.

The following data types are defined. Note that these are all defined as `Float`. Change as needed for your application

Length types. Prefix is “len”. Base unit is meters.

```
type len_m is new Float;   -- length in meters
type len_ft is new Float;  -- length in feet
type len_A is new Float;   -- length in Angstroms
```

Area types. Prefix is “area”. Base unit is meters^2.

```
type area_m2 is new Float;  -- area in square meters
```

Volume types. Prefix is “vol”. Base unit is liters.

```
type vol_l is new Float;   -- volume in liters
type vol_m3 is new Float;  -- volume in cubic meters
```

Mass types. Prefix is “mass”. Base unit is kilograms.

```
type mass_kg is new Float; -- mass in kilograms
type mass_lb is new Float; -- mass in pounds

Force types.  Prefix is ``force''.  Base unit is newtons.
type force_n is new Float; -- force in newtons
```

Temperature types. Prefix is "temp". Base unit is celsius. Note that range limits are not given for temperatures since the difference between two temperatures may exceed the range.

```
type temp_k is new Float; -- temperature in kelvin
type temp_c is new Float; -- temperature in celsius
type temp_f is new Float; -- temperature in Fahrenheit
```

Pressure types. Prefix is "press". Base unit is pascal.

```
type press_p is new Float;    -- pressure in pascals
type press_mb is new Float;   -- pressure in millibars
type press_atm is new Float;  -- pressure in atmospheres
type press_inHg is new Float; -- pressure in inches of mercury
```

Velocity types. Prefix is "vel". Base unit is m/s.

```
type vel_m_s is new Float;   -- velocity in meters/second
type vel_mph is new Float;   -- velocity in miles per hour
type vel_km_h is new Float;  -- velocity in kilometers/hour
type vel_knots is new Float; -- velocity in knots
```

Acceleration types. Prefix is "accel". Base unit is m/(s^2).

```
type accel_m_s2 is new Float; -- acceleration in meters per second squared
type accel_g is new Float;    -- acceleration in units of Earth gravity
```

Angular type. Prefix is "ang". Base unit is radians.

```
type ang_r is new Float; -- angle in radians
type ang_d is new Float; -- angle in degrees
```

Rotation rate types. Prefix := "rot". Base unit is radians/second.

```
type rot_r_s is new Float; -- rotation in radians per second
type rot_d_s is new Float; -- rotation in degrees per second
```

Magnetic types. Prefix is "mag". Base unit is Gauss.

```
type mag_g is new Float; -- magnetic field in gauss
```

Electromotive force types. Prefix is "emf". Base unit is Volt.

```
type emf_v is new Float; -- electromotive force in volts
```

Electrical current types. Prefix is "curr". Base unit is Amper.

```
type curr_a is new Float; -- electrical current in amps
```

Electrical resistance types. Prefix is "res". Base unit is Ohms.

```
type res_o is new Float; -- electrical resistance in ohms
```

Frequency types. Prefix is "freq". Base unit is Hertz. Time types. Prefix is "time". Base unit is Seconds.

Note that Ada has a predefined Duration type that is a fixed point type Seconds is defined as a subtype of this. The other times (minutes and hours) are derivative types so as to maintain similar precision. If needed, they could be changed to Float or something else.

```
type freq_hz is new Float;     -- frequency in Hertz
subtype time_s is Duration;    -- time in seconds
-- (use subtype because seconds is identical to duration)
type time_m is new Duration;  -- time in minutes
type time_h is new Duration;  -- time in hours
```

The following type conversions are defined (add any other conversions needed for your application):

```
function to_feet(dist : len_m) return len_ft;
function to_Angstroms(dist : len_m) return len_A;
function to_meters(dist : len_ft) return len_m;
function to_meters(dist : len_A) return len_m;
function to_liters(vol : vol_m3) return vol_l;
function to_meters3(vol : vol_l) return vol_m3;
function to_pounds(mass : mass_kg) return mass_lb;
function to_kilograms(mass : mass_lb) return mass_kg;
function to_Farenheit(temp : temp_c) return temp_f;
function to_Kelvin(temp : temp_c) return temp_k;
function to_Celsius(temp : temp_f) return temp_c;
function to_Celsius(temp : temp_k) return temp_c;
function to_milliBar(pressure : press_p) return press_mb;
function to_Atmosphere(pressure : press_p) return press_atml;
function to_inHg(pressure : press_p) return press_inHgl;
function to_Pascal(pressure : press_mb) return press_p;
function to_Pascal(pressure : press_atm) return press_p
function to_Pascal(pressure : press_inHg) return press_p;
function to_mph(vel : vel_m_s) return vel_mph;
function to_km_h(vel : vel_m_s) return vel_km_h;
function to_knots(vel : vel_m_s) return vel_knots;
function to_m_s(vel : vel_knots) return vel_m_s;
function to_m_s(vel : vel_km_h) return vel_m_s;
function to_m_s(vel : vel_mph) return vel_m_s;
function to_m_s2(accel : accel_g) return accel_m_s2;
function to_g(accel : accel_m_s2) return accel_g;
function to_degrees(ang : ang_r) return ang_d;
function to_radians(ang : ang_d) return ang_r;
function to_r_s(rot : rot_d_s) return rot_r_s;
function to_d_s(rot : rot_r_s) return rot_d_s;
function to_hz(period : time_s) return freq_hz;
function to_minutes(period : time_s) return time_m;
function to_hours(period : time_s) return time_h;
function to_seconds(freq : freq_hz) return time_s
     with Global => null, pre => (freq /= 0.0);
function to_seconds(period : time_m) return time_s;
function to_seconds(period : time_h) return time_s;
```

By default, Ada generally expects the result of an operation to have the same type as arguments to that operation (eg. `integer*integer` gives `integer`). This is not what is desired when multiplying and dividing physical units. Therefor, the following operations are defined. Note that this is just enough to get one started. Many more combinations can be defined as needed.

```
function ”/”(Left : len_m; Right : Duration) return vel_m_s
     with Global => null, pre => (Right /= 0.0);
function ”*”(Left, Right : len_m) return area_m2;
function ”*”(Left : len_m; Right : area_m2) return vol_m3;
function ”*”(Left : area_m2; Right : len_m) return vol_m3;
function ”*”(Left : mass_kg; Right : accel_m_s2) return force_n;
function ”*”(Left : accel_m_s2; Right : mass_kg) return force_n;
function ”/”(Left : force_n; Right : accel_m_s2) return mass_kg
     with Global => null, pre => (Right /= 0.0);
function ”/”(Left : force_n; Right : mass_kg) return accel_m_s2
     with Global => null, pre => (Right /= 0.0);
function ”*”(Left : vel_m_s; Right : Duration) return len_m;
function ”*”(Left : Duration; Right : vel_m_s) return len_m;
function ”/”(Left : vel_m_s; Right : Duration) return accel_m_s2
     with Global => null, pre => (Right /= 0.0);
function ”*”(Left : accel_m_s2; Right : Duration) return vel_m_s;
function ”*”(Left : Duration; Right : accel_m_s2) return vel_m_s;
function ”*”(Left : rot_d_s; Right : Duration) return ang_d;
function ”*”(Left : Duration; Right : rot_d_s) return ang_d;
function ”*”(Left : curr_a; Right : res_o) return emf_v;
function ”*”(Left : res_o; Right : curr_a) return emf_v;
function ”/”(Left : emf_v; Right : curr_a) return res_o
     with Global => null, pre => (Right /= 0.0);
function ”/”(Left : emf_v; Right : res_o) return curr_a
     with Global => null, pre => (Right /= 0.0);
```