

# 1 Currency data type

```
"goi/money.rb" 1a ≡
# Monetary Data Type
<License 1c>
<Required Modules 1d>
module Goi
  <Goi classes 2a, ... >
  module_function
  <Goi functions ?>
end
◇
```

```
"money_test.rb" 1b ≡

<License 1c>
require 'goi/money.rb'
include Goi
<Tests 7b>
◇
```

```
<License 1c> ≡
# Copyright (C) 2003–2005, Javier Goizueta <javier@goizueta.info>
#
# This program is free software; you can redistribute it and/or
# modify it under the terms of the GNU General Public License
# as published by the Free Software Foundation; either version 2
# of the License, or (at your option) any later version.
◇
```

Macro referenced in [1ab](#).

## 1.1 Currencies

This class represents a monetary currency, with the purpose of holding its attributes (names, abbreviations, codes, number of decimals used) and converting between different currencies (using fixed rates).

```
<Required Modules 1d> ≡
require 'goi/nfmt'
require 'goi/tools'
◇
```

Macro referenced in [1a](#).

```

< Goi classes 2a > ≡
class Currency
  include StateEquivalent
  def initialize (name,abr,abr_pl,symbol,codeA,codeN,ndec,ref_rate,def_lang=:en)
    @name=name
    @abr = abr
    @abr_pl = abr_pl
    @symbol = symbol
    @codeA = codeA
    @codeN = codeN
    @ndec = ndec
    @ref_rate = ref_rate
    @def_lang = def_lang
  end
  attr_reader :name, :abr, :abr_pl, :symbol, :codeA, :codeN, :ndec, :ref_rate, :def_lang
  attr_writer :def_lang, :ndec, :ref_rate

  def convert(v,dest_curr)
    v *= dest_curr.ref_rate
    v /= @_@ref_rate
  end
end
◇

```

Macro defined by [2ab](#), [3ef](#), [7a](#).

Macro referenced in [1a](#).

Defines: [Currency 2b](#), [3abef](#), [4a](#), [7b](#).

We'll also keep a repository of common currencies.

```

< Goi classes 2b > ≡

class Currency
  @@currs = {
    < Predefined Currencies 3c, ... >
    :ref=>< euro 3a >, # reference : euro
    :def=>< euro 3a > # default : currency: reference
  }
  def self.define(tag,curr_def)
    @@currs[tag]=curr_def
  end
  def self.get(tag)
    @@currs[tag]
  end
end
◇

```

Macro defined by [2ab](#), [3ef](#), [7a](#).

Macro referenced in [1a](#).

Uses: [Currency 2a](#).

To reduce program verbosity, programs can set default currency values; for example for an application working mainly with euros and formatting according to the spanish language conventions, this can be done:

```

eur = Currency.get(:eur)
pts = Currency.get(:esp)

# set euro as default currency
Currency.define(:def, eur)
# set default language to spanish for euro
eur.def_lang = :es

```

And we could define special default formatting options for a currency:

```
eur.define_defaultFmt eur.defaultFmt.sep(' ','.',[3])
```

```
< euro 3a > ≡
  Currency.new('Euro','euro','euros',"x80",'EUR','978',2,1)◇
```

Macro referenced in [2b](#), [3c](#).

Uses: [Currency 2a](#).

```
< peseta 3b > ≡
  Currency.new('Peseta','Pts.','Pts.',' ','ESP','724',0,166.386,:es)◇
```

Macro referenced in [3d](#).

Uses: [Currency 2a](#).

```
< Predefined Currencies 3c > ≡
  :eur=>< euro 3a >,◇
```

Macro defined by [3cd](#).

Macro referenced in [2b](#).

```
< Predefined Currencies 3d > ≡
  :esp=>< peseta 3b >,◇
```

Macro defined by [3cd](#).

Macro referenced in [2b](#).

```
< Goi classes 3e > ≡
class Currency
  @@def_fmts = {}
  def defaultFmt
    fmt = @@def_fmts[@codeA]
    fmt = NFmt.get(@def_lang).dup.prec(@ndec,:fix,inf) if fmt==nil
    fmt
  end
  def define_defaultFmt(fmt)
    @@def_fmts[@codeA] = fmt
  end
end
◇
```

Macro defined by [2ab](#), [3ef](#), [7a](#).

Macro referenced in [1a](#).

Uses: [Currency 2a](#).

## 1.2 Money

Class `Money` will represent amounts of money. We will define arithmetic operations between money and numbers. Money objects can optionally round after each operation done with them (except dividing by another money amount, which yields a simple number).

```
< Goi classes 3f > ≡

class Money
  include StateEquivalent
  @@num_t = Float
  def initialize (v=0,curr=Currency.get(:def),fmt=nil,auto_round=true)
    set! v, curr, fmt, auto_round
  end
  < Money Constructors 4a >

  < Money Attributes 5b >

  def convert(curr,fmt=nil)
    Money.new(@curr.convert(@v,curr),curr,fmt,@autor) # dup.convert!(curr,fmt)
  end
end
```

```

def writeFmt()
  # to do: print currency optionally and handle negative amount format
  @v.writeFmt(@fmt)
end
def readFmt!(txt)
  @v = @@num_t.readFmt(txt, @fmt)
  < Modified Money 6b >
end
def to_s; writeFmt; end
def round
  Money.new(@v.roundFmt(@fmt),@curr,@fmt,@autor) # dup.round!
end
< Money Arithmetic 4b >
< Money Comparison 5a >
< Money Mutate Methods 6c, ... >
protected
< Money Protected 6a >
end
◇

```

Macro defined by [2ab](#), [3ef](#), [7a](#).

Macro referenced in [1a](#).

Defines: [Money 4ab](#), [7ab](#).

Uses: [Currency 2a](#).

```

< Money Constructors 4a > ≡
def Money.formatted(v,fmt,curr=Currency.get(:def),auto_round=true)
  Money.new v, curr, fmt, auto_round
end
def Money.rounding(v,curr=Currency.get(:def),fmt=nil)
  Money.new v, curr, fmt, true
end
def Money.nonrounding(v,curr=Currency.get(:def),fmt=nil)
  Money.new v, curr, fmt, false
end
◇

```

Macro referenced in [3f](#).

Uses: [Currency 2a](#), [Money 3f](#).

```

< Money Arithmetic 4b > ≡
def *(x)
  Money.new(@v*x,@curr,@fmt,@autor)
end
def +(m)
  Money.new(@v+m.amount(@curr),@curr,@fmt,@autor)
end
def -(m)
  Money.new(@v-m.amount(@curr),@curr,@fmt,@autor)
end
def / (xm)
  if xm.is_a?(Money)
    @v/xm.amount (@curr)
  else
    Money.new(@v/xm, @curr, @fmt, @autor)
  end
end
def -@
  Money.new(-@v,@curr,@fmt,@autor)
end
◇

```

Macro referenced in [3f](#).

Uses: [Money 3f](#).

When comparing `Money` values of different currencies, we'll use the rounding mode of both values to decide whether to perform a rounded comparison or not: only if both values use automatic rounding, the comparison will be done on rounded converted values. When rounded comparison is performed, the resulting relation is not an order-relation for different currency values; it is not reflexive, since conversion and rounding is done based on the first operand. As this may lead to confusion, I've decided to not do rounded comparison for different currencies; if it is needed explicitly converting the currency will achieve it.

```

<Money Comparison 5a> ≡
  def <=>(m2)
    return nil unless m2
    v1 = @v
    v2 = m2.amount(currency)
    # if auto_round && m2.auto_round && (currency != m2.currency)
    #   v2 = v2.roundFmt(@fmt)
    # end
    v1 <=> v2
  end
  include Comparable
  ◇

```

Macro referenced in [3f](#).

The `amount` attribute is the only part of the public interface that reveals the underlying type used for money amounts. Perhaps it should be make protected. Currently it is only used by `<=>`.

```

<Money Attributes 5b> ≡
  def amount(curr=nil)
    if curr!=nil && curr!=@curr
      return @curr.convert(@v,curr)
    end
    @v
  end
  def currency
    @curr
  end
  def format
    @fmt
  end
  def auto_round
    @autor
  end
  ◇

```

Macro referenced in [3f](#).

### 1.2.1 Mutating Methods

If `BigDecimal` is used as the base type, we cannot use a floating point value to initialize a `Money` because there's no conversion from `Float` to `BigDecimal`.

Also, if the base type is no `BigDecimal` we cannot use a `BigDecimal` for initialization, because `BigDecimal` defines a `prec` method for other uses.

A value formatted as text is always safe for initialization.

Note also that if the base type is `BigDecimal` we cannot multiply or divide `Money` by `Float` values; we must use either integers or big-decimals, because the big-decimal will be coerced into a floating point value for the operation, and there's no conversion from floating point to big-decimal.

```

<Money Protected 6a> ≡
  def set!(v,curr,fmt,auto_round)
    @curr = curr
    @fmt = fmt
    @fmt = curr.defaultFmt if @fmt==nil
    @autor = auto_round

    if v.is_a? String
      @v = @@num_t.readFmt(v, @fmt)
    elsif v.is_a? @@num_t
      # used internally (arithmetic operators)
      @v = v
    else
      # @v = v.prec(@@num_t) # unsafe: does not work for all types
      if auto_round
        @v = @@num_t.readFmt(v.writeFmt(@fmt),@fmt) # rounds
      else
        @v = NFmt.convert(v, @@num_t)
      end
    end
  end
  <Modified Money 6b>
end
◇

```

Macro referenced in [3f](#).

When a money object is modified, it should be rounded if necessary.

```

<Modified Money 6b> ≡
  if @autor
    @v = @v.roundFmt(@fmt)
  end
◇

```

Macro referenced in [3f](#), [6a](#).

```

<Money Mutate Methods 6c> ≡
  def convert!(curr,fmt=nil)
    set! @curr.convert(@v,curr),curr,fmt,@autor
  self
  end
◇

```

Macro defined by [6cd](#).

Macro referenced in [3f](#).

```

<Money Mutate Methods 6d> ≡
  def round!
    set! @v.roundFmt(@fmt),@curr,@fmt,@autor
  self
  end
◇

```

Macro defined by [6cd](#).

Macro referenced in [3f](#).

I'll provide an interface to change the base type. The base type is in a class variable, and the effects of changing it have not been thoroughly looked at, so it'd be better not to mix `Money` objects created with different base types in calculations.

⟨ *Goi classes 7a* ⟩ ≡

```
class Money
  def self.setNumType(nt)
    @@num_t = nt
  end
  def self.getNumType()
    @@num_t
  end
end
end
◇
```

Macro defined by [2ab](#), [3ef](#), [7a](#).

Macro referenced in [1a](#).

Uses: [Money 3f](#).

## 2 Tests

⟨ *Tests 7b* ⟩ ≡

```
# TESTS
def test ()
```

```
  eur_fmt = NFmt.sep(' ',' ',' ',[3]).prec(2, :fix, :inf)
  esp_fmt = NFmt.sep(' ',' ',' ',[3]).prec(0, :fix, :inf)
```

```
  x = Money.new(123.23, Currency.get(:eur), eur_fmt)
  p x.writeFmt
  x *= 2.113322
  p x.writeFmt
  y = x.convert(Currency.get(:esp), esp_fmt)
  p y.writeFmt
  z = Money.new(1, Currency.get(:esp), esp_fmt)
  z += x
  p z.writeFmt
```

```
end
```

```
test
```

```
◇
```

Macro referenced in [1b](#).

Uses: [Currency 2a](#), [Money 3f](#).

## 3 Índices

### 3.1 Archivos

"[goi/money.rb](#)" Defined by [1a](#).

"[money\\_test.rb](#)" Defined by [1b](#).

### 3.2 Fragmentos

⟨ *Goi classes 2ab, 3ef, 7a* ⟩ Referenced in [1a](#).

⟨ *Goi functions ?* ⟩ Referenced in [1a](#).

⟨ *License 1c* ⟩ Referenced in [1ab](#).

⟨ *Modified Money 6b* ⟩ Referenced in [3f](#), [6a](#).

⟨ *Money Arithmetic 4b* ⟩ Referenced in [3f](#).

⟨ *Money Attributes 5b* ⟩ Referenced in [3f](#).

⟨ *Money Comparison 5a* ⟩ Referenced in [3f](#).

⟨ *Money Constructors 4a* ⟩ Referenced in [3f](#).

⟨ *Money Mutate Methods 6cd* ⟩ Referenced in [3f](#).

⟨ Money Protected [6a](#) ⟩ Referenced in [3f](#).  
⟨ Predefined Currencies [3cd](#) ⟩ Referenced in [2b](#).  
⟨ Required Modules [1d](#) ⟩ Referenced in [1a](#).  
⟨ Tests [7b](#) ⟩ Referenced in [1b](#).  
⟨ euro [3a](#) ⟩ Referenced in [2b](#), [3c](#).  
⟨ peseta [3b](#) ⟩ Referenced in [3d](#).

### 3.3 Identificadores

Currency: [2a](#), [2b](#), [3abef](#), [4a](#), [7b](#).

Money: [3f](#), [4ab](#), [7ab](#).