GFR-2
The Gear Standard Runtime Library

Version 0.1

Markéta Lisová

June 27, 2015

Contents

Preface

The *Gear Standard Runtime Library* (the GSRL[1], not to be confused with GLS, which is the Gear Language Specification, GFR-0) is a specification for a set of Gear modules, types and classes that come shipped together with every GVM (Gear Virtual Machine, GFR-1).

Some features of the CSL are not available on every platform/CVM combination, and these differences are documented in this document, along with advices on how to properly query for their presence or absence.

Many functions that are a part of the CSL are implemented natively in particular CVMs, mostly because they use the low-level APIs provided by a CVM, or simply for performance reasons, and therefore it is not possible to list their source in this specification, only function and method signatures.

Status of This GFR

This GFR is active and mandatory for every proper Gear implementation, without exceptions. The status of this GFR is not likely to change.

Some particular components defined in this GFR are only optional, as specified, and need not to be implemented in a proper Gear implementation.

[1]Or, as a mnemonic, GSRLib.

Chapter 1

The Language Module

The `Gear/Language` module is implicitly imported into every Gear source, and there is no way to opt-out of this behaviour.

For brevity, we will omit the "`Gear/`" in this and every following chapter, and unless specified otherwise, it will be implicitly added to every occurrence of any CSL module as vendor specifier.

Every Gear source can be viewed as starting with the following code:

```
use Gear/Language.{_}
```

The `Language` module is designed to be as much not intrusive as possible, e.g. the `Language.Object` does not define any instance methods.[1] To allow users to use some typical methods querying `Object` instances, Gear offers attribute syntax along with those methods predefined in the `Language.Predef` object (§1.10).

1.1 The Any Type

```
module Language

type Any := ;; implementation-defined, blank slate
```

1.2 The Object Class

```
module Language

class Object extends Any
```

[1]This is very much unlike e.g. Java, whose `java.lang.Object` is pre-flooded with methods.

```
end class

object Object
begin
  message new (params: Variadic_Arguments): self.type
  end message

  constructor ()
  end constructor

  clone ()
  end clone
end object
```

1.3 The Nothing & Undefined Classes

```
module Language

immutable sealed class Nothing extends /* every type */
end class

immutable sealed class Undefined extends Nothing
end class
```

1.4 The Unit Class

```
module Language

immutable sealed class Unit extends Object
end class
```

1.5 The Type Class

```
module Language

abstract class Type [T] extends Object
end class

object Type
begin
end object
```

1.6 The ‘Class’ Class

```
module Language

class Class [T] extends Type[T]
end class

object Class
begin
  message new (name: Symbol, &init: Class[_] -> Unit): Class[_]
  end message
end object
```

1.7 The Metaclass Class

```
module Language

class Metaclass [T <: Class[T]] extends Type[T]
end class

object Metaclass
begin
end object
```

1.8 The Magnitude Class

```
module Language

abstract class Magnitude extends Object
end class

object Magnitude
begin
end object
```

1.9 The Number Class & Number_Like Trait

```
module Language

trait Number_Like extends Magnitude
  constraint value as Number_Like
  end constraint

  message as_number (): Number
  end message
end trait

abstract sealed class Number
    extends Magnitude
       with Number_Like
end class

object Number
begin
  type Integer := ;; implementation-defined
  end type
  type Integer_Unsigned := ;; implementation-defined
    ;; could be: Integer with constraint { value >= 0 }
  end type
  type Real := ;; implementation-defined
  end type
  type Decimal := ;; implementation-defined
  end type
  type Decimal_Unsigned := ;; implementation-defined
    ;; could be: Decimal with constraint { value >= 0 }
  end type
  type Fixed_Point_Number [
      Delta  <: Literal_Singleton_Type[Decimal_Unsigned],
      Digits <: Literal_Singleton_Type[Integer_Unsigned],
      Range  <: (Literal_Singleton_Type[Decimal],
                 Literal_Singleton_Type[Decimal],
                 Literal_Singleton_Type[Boolean])]
      := ;; implementation-defined
  end type
  immutable class Rational (
      val numerator: Number_Like,
      val denominator: Number_Like with constraint { value /= 0 })
      extends Number
  end class
  immutable class Complex (
      val real: Number_Like,
      val imaginary: Number_Like) extends Number
  end class

  operator =    (other: Number_Like) end operator
  operator /=   (other: Number_Like) end operator
  operator >    (other: Number_Like) end operator
```

```
   operator >=   (other: Number_Like) end operator
   operator <    (other: Number_Like) end operator
   operator <=   (other: Number_Like) end operator
   operator <>   (other: Number_Like) end operator
   operator <=>  (other: Number_Like) end operator
   operator <>=  (other: Number_Like) end operator
   operator /<>= (other: Number_Like) end operator
   operator /<>  (other: Number_Like) end operator
   operator /<=  (other: Number_Like) end operator
   operator /<   (other: Number_Like) end operator
   operator />=  (other: Number_Like) end operator
   operator />   (other: Number_Like) end operator
 end object
```

1.9.0.1 Standard Number Operators

Comparison operators. The following table shows a matrix of comparison operators on `Number`. "Unordered" means that either or both of the operands is a `Number.Not_a_Number`. The results described are relative to the number the operator is applied to (defined by binding direction of the operator). "Raises" means that if either or both of the operands is a `Number.Not_a_Number`, then an error is raised.

Table 1.1: Number comparison operators

Op.	Greater	Less	Equal	Unordered	Raises	Relation
=	no	no	yes	no	no	equal
/=	yes	yes	no	yes	no	unordered, less, or greater
>	yes	no	no	no	yes	greater
>=	yes	no	yes	no	yes	greater, or equal
<	no	yes	no	no	yes	less
<=	no	yes	yes	no	yes	less, or equal
<>	yes	yes	no	no	yes	less, or greater
<>=	yes	yes	yes	no	yes	less, greater, or equal
/<>=	no	no	no	yes	no	unordered
/<>	no	no	yes	yes	no	unordered, or equal
/<=	yes	no	no	yes	no	unordered, or greater
/<	yes	no	yes	yes	no	unordered, equal, or greater
/>=	no	yes	no	yes	no	unordered, or less
/>	no	yes	yes	yes	no	unordered, less, or equal

1.10 The Language.Predef Object