8 INDEX

[This Index is not part of the Standard.]

8.1

This Index indicates the page of definition of every class and feature appearing in the Required Flatshort Forms of section 5

8.2

Following the standard Eiffel conventions, feature names appear in lower-case italics and class names, when making up index entries, in UPPER-CASE ITALICS. Operator functions appear under ***prefix*** and ***infix***; for example division appears under ***infix*** "/". This also applies to boolean operators, which appear under ***infix*** "***and***", ***infix*** "***and then***" and so on.

8.3

In a class entry, the class appears in *UPPER-CASE ITALICS*. Each reference to a feature name is followed by the name of the class or classes in which it is available, each with the corresponding page. To avoid any confusion with occurrences of the class name in its other role – as an index entry pointing to the beginning of the class specification – the class name in this case appears in UPPER-CASE ROMAN.

read*character, read*double, read*integer, read*line, read*real, read*stream, read*word, **to_**next*line. (See 5.14, page 38, and 5.15, page 39.)

7.17 • Addition to *EXCEPTIONS* of a procedure *die* that terminates the execution cleanly with a given exit status, without triggering an exception. (See page 44.)

7.18 • In class *STRING*, replacement of the *adapt* function by a more convenient procedure *make_from_string* which descendants of the class can use to initialize a string-like object from a manifest string, as in !! *t.make_from_string* (*"THIS STRING"*), where the type of *t* is a descendant of *STRING*. (See page 34.)

7.19 • Similarly, addition to *ARRAY* of a procedure *make_from_array* allowing initialization from a manifest array, as in !! *a.make_from_array* (<<*a, b, c, d*>>).

7.20 • Removal from *STRING* of a number of features which some committee members judged too specialized: *mirror*, *mirrored*, *share*, *shared_with*, *item_code*, *has*, *prepend*, *set*, *prune*, *prune_all*. Renaming of *replace_substring* to *put_substring*. Removal of *fill_blanks*, replaced by *fill* (applying to an arbitrary character). Change of the result type of *out* to *STRING* (rather than ***like*** *Current*).

7.4 • Addition of a number of assertion clauses to the features of *GENERAL*, in particular to specify more precisely the semantics of equality, copying, cloning and conformance.

7.5 • Addition to *GENERAL* of a function *stripped* such that *stripped* (*a*) is a clone of the current object limited to the fields that apply to *a*'s dynamic type. As a result, the old semantics of copying and equality mentioned in 7.2 may now be achieved through calls such as *a*.*copy* (*b*.*stripped* (*a*)) and *equal* (*a, b*.*stripped* (*a*)). (See page 12.)

7.6 • Addition to *GENERAL* of *object_id* and *id_object* to allow unique identification of objects. (See page 12.)

7.7 • In class *PLATFORM*, removal of the assumption that *Character_bits*, *Integer_bits*,*Real_bits* and *Double_bits* are constants. This does notintroduce any incompatibility with earlier uses except if they relied onthe specific numerical values. (See page 46.)

7.8 • Removal of *PLATFORM* from the universal inheritance hierarchy; *PLATFORM* is no longer a parent of *ANY* and hence an ancestor of every class, and has no particular language-defined role; classes that need its facilities must name it explicitly among their proper ancestors. This is actually a language change. (See section 4, page 10.)

7.9 • Addition to *PLATFORM* of features *Maximum_integer*, *Minimum_integer*, *Maximum_character_code* and *Minimum_character_code*. (See page 46.)

7.10 • Addition to *COMPARABLE* of *min* and *max* functions and of a three-way comparison function, *three_way_comparison*, which returns 0, –1 or 1. (See page 15.)

7.11 • Addition to the arithmetic basic classes of functions *abs* and *sign* (the latter defined in terms of *three_way_comparison*). Addition to *REAL* and *DOUBLE* of *floor*, *ceiling*, *rounded* and *integer_part*. Addition to *DOUBLE* of *real_part*. (See page 23 and following.)

7.12 • Addition of inheritance links making all basic classes (*INTEGER* and so on) heirs of *HASHABLE*, so that it is now possible to hash any object. (See section 4, page 10.) Removal of function *is_hashable* and the corresponding preconditions.

7.13 • Addition to *ARRAY* of features *enter* and *entry* as redefinable synonyms to *put* and *item* (or ***infix*** "@"), the latter becoming frozen. (See page 32.)

7.14 • Addition to *STORABLE* of a procedure *independent_store* which produces machine-independent representations of object structures. (See page 42.)

7.15 • Addition of a few features to class *FILE* describing file opening and opening modes (such as read-only or read-write). In earlier presentations the corresponding class was *UNIX_FILE*. The new class is very similar but removes any Unix-specific aspect. (See 5.15, page 39.)

7.16 • Changes of names in class *STD_FILES* and *FILE* : for consistency with the usual Eiffel naming style, underscores were added and abbrevations were expanded. In the following list (which uses the order of appearance of the features in *STD_FILES*), the added underscores appear as * and the added letters appear in ***bold italics***: *last*char**acter**, last*double, last*real, last*int**eger**, last*string, put*bool**ean**, put*char**acter**, put*double, put*int**eger**, **put_**new*line, put*real, put*string,*

6.5 Intermediate corrections

During the time when a vintage is in effect, minor corrections may prove necessary, due for example to typographical errors in the current version of this Standard or to inconsistencies discovered by users or implementors of Eiffel. In such a case the chairman of the Library Committee of NICE may, at his discretion, submit a motion covering one or more revisions. To be approved, such motions shall require a unanimous vote of the Library Committee, with the possible exception of any member who has notified the chairman of an absence of more than one month. If approved, such a revision shall receive a revision level and shall give rise to a modified Kernel Library Standard, identified as "Vintage *year* Level *revision_level*". The modifications shall be integrated into the following year's vintage.

6.6 Eiffel Kernel Supplier requirements

Any provider of an Eiffel environment must make the following information available to any NICE member:

6.7 • Vintage and revision level currently supported.

6.8 • Any features not supported. (It is not permitted to have a non-supported class.)

6.9 • List of classes needed by kernel classes, but not in the kernel, hereafter referred to as para-kernel classes.

6.10 • Full inheritance hierarchy of kernel and para-kernel classes.

6.11 • List of names of features (immediate or inherited) that appear in the provider's kernel classes but not in this Standard.

7 APPENDIX B: DIFFERENCES

[This Appendix is not part of the Standard.]

The following differences exist between this Standard and earlierpresentations of the Kernel Library:

7.1 • Addition to *GENERAL* of a query *default* which returns the default value of the type of the current object. This also addresses the need to obtain the default value for type *POINTER* ; for convenience, since *POINTER* has no manifest constant, a query *default_pointer* has also been included. (See page 13.)

7.2 • Adaptation of the semantics of *copy* and equality features (*equal*, *is_equal* and their *standard_* versions) so that the result is true if and only if the objects are truly identical, and in particular have the same type. This implies a language change too; the previous definition was non-symmetric so that *a*.*copy* (*b*) and *equal* (*a, b*) only applied to the fields corresponding to the attributes of *a*'s type. The earlier effect can still be achieved through function *stripped,* as explained next in 7.5. (See page 12.)

7.3 • Addition to *GENERAL* of a frozen feature *same_type* which specifies conformance both ways. Addition of the requirement that *conforms_to* is frozen too. (See page 12.)

6 APPENDIX A: THE KERNEL STANDARDIZATION PROCESS

[This Appendix is not part of the Standard.]

6.1 Why plan a process?

The Eiffel Kernel Library cannot be specified for eternity. Ideas willcome up for new classes and features; ways will be found to do thingsbetter. The evolution process must be fast enough to enable Eiffel usersto benefit from this flow of ideas and avoid technical obsolescence, but orderly enough to protect their existing investments and modes of operation.

6.2 Cycle time

A revision every ten to fifteen years, as has occurred for programming language standards (Fortran, C and Ada are examples) is not appropriate for the Eiffel Kernel Library. It would foster complacency most of the time, and major upheavals when a revision is finally brought into effect. A yearly process, similar to the upgrading of wines, car models and stable software products, provides the right pace of change.

6.3 Vintages

Each revision of this Standard describes a **vintage** of theEiffel Library Kernel Standard. The present version is vintage 1995.

6.4 Yearly schedule

The following deadlines apply to year *year*:

6.4.1 • 1 January: Vintage *year* takes effect.

6.4.2 • 1 April: first permitted date for starting discussions on Vintage *year+1* in NICE's Library Committee. (1 January to 31 March is acooling-off period.)

6.4.3 • 1 May: first permitted date for submitting formal proposals to the Library Committee for Vintage *year + 1*.

6.4.4 • 1 July: last permitted date for submitting initial proposals for Vintage *year + 1*.

6.4.5 • 1 September: last permitted date for submitting final proposals (which may result from merging of several proposals) for Vintage *year + 1*.

6.4.6 • 1 October: last date by which the Committee may have defined Vintage *year +1*.

This schedule is applicable starting with vintage 96. For the present vintage (95), the first, the date of applicability is 1 July 1995.

5.26 Class *REAL_REF*

indexing

description: *"Reference class for REAL"*

class interface

REAL_REF

feature -- Access

item: *REAL*
-- Real value

hash_code: *INTEGER*
-- Hash code value
-- (From *HASHABLE*.)
ensure
good_hash_value: *Result >= 0*

feature -- Element change

set_item (*r*: *REAL*)
-- Make *r* the associated real value.
ensure
item_set: *item = r*

end

5.25 Class *POINTER_REF*

indexing

description: *"Reference class for POINTER"*

class interface

POINTER_REF

feature -- Access

item: *POINTER*
-- Pointer value

hash_code: *INTEGER*
-- Hash code value
-- (From *HASHABLE*.)

ensure
good_hash_value: *Result >= 0*

feature -- Element change

set_item (*p*: *POINTER*)
-- Make *p* the associated pointer value.

ensure
item_set: *item = p*

end

5.24 Class *INTEGER_REF*

indexing

description: *"Reference class for INTEGER"*

class interface

INTEGER_REF

feature -- Access

item: *INTEGER*
-- Integer value

hash_code: *INTEGER*
-- Hash code value
-- (From *HASHABLE*.)
ensure
good_hash_value: *Result* >= *0*

feature -- Element change

set_item (*i*: *INTEGER*)
-- Make *i* the associated integer value.
ensure
item_set: *item* = *i*

end

5.23 Class *DOUBLE_REF*

indexing

description: *"Reference class for DOUBLE"*

class interface

DOUBLE_REF

feature -- Access

item: *DOUBLE*
-- Double value

hash_code: *INTEGER*
-- Hash code value
-- (From *HASHABLE*.)
ensure
good_hash_value: *Result >= 0*

feature -- Element change

set_item (*d*: *DOUBLE*)
-- Make *d* the associated double value.
ensure
item_set: *item = d*

end

5.22 Class *CHARACTER_REF*

```
indexing
    description: "Reference class for CHARACTER"
class interface
    CHARACTER_REF
feature -- Access
    item: CHARACTER
            -- Character value
    hash_code: INTEGER
            -- Hash code value
            -- (From HASHABLE.)
        ensure
            good_hash_value: Result >= 0
feature -- Element change
    set_item (c: CHARACTER)
            -- Make c the associated character value.
        ensure
            item_set: item = c
end
```

5.21 Class *BOOLEAN_REF*

indexing

description: *"Reference class for BOOLEAN"*

class interface

BOOLEAN_REF

feature -- Access

item: *BOOLEAN*
-- Boolean value

hash_code: *INTEGER*
-- Hash code value
-- (From *HASHABLE*.)
ensure
good_hash_value: *Result >= 0*

feature -- Element change

set_item (*b*: *BOOLEAN*)
-- Make *b* the associated boolean value.
ensure
item_set: *item = b*

end

5.20 Class *PLATFORM*

```
indexing
    description: "Platform–dependent properties. This class may be
        used as ancestor by classes needing its facilities"
class interface
    PLATFORM
feature -- Access
    Boolean_bits: INTEGER
            -- Number of bits in a value of type BOOLEAN
        ensure
            meaningful: Result >= 1
    Character_bits: INTEGER
            -- Number of bits in a value of type CHARACTER
        ensure
            meaningful: Result >= 1
            large_enough: 2 ^ Result >= Maximum_character_code
    Double_bits: INTEGER
            -- Number of bits in a value of type DOUBLE
        ensure
            meaningful: Result >= 1;
            meaningful: Result >= Real_bits
    Integer_bits: INTEGER
            -- Number of bits in a value of type INTEGER
        ensure
            meaningful: Result >= 1;
            large_enough: 2 ^ Result >= Maximum_integer;
            large_enough_for_negative: 2 ^ Result >= – Minimum_
                integer
    Maximum_character_code: INTEGER
            -- Largest supported code for CHARACTER values
        ensure
            meaningful: Result >= 127
    Maximum_integer: INTEGER
            -- Largest supported value of type INTEGER.
        ensure
            meaningful: Result >= 0
    Minimum_character_code: INTEGER
            -- Smallest supported code for CHARACTER values
        ensure
            meaningful: Result <= 0
    Minimum_integer: INTEGER
            -- Smallest supported value of type INTEGER
        ensure
            meaningful: Result <= 0
    Pointer_bits: INTEGER
            -- Number of bits in a value of type POINTER
        ensure
            meaningful: Result >= 1
    Real_bits: INTEGER
            -- Number of bits in a value of type REAL
        ensure
            meaningful: Result >= 1
end
```

5.19 Class *ARGUMENTS*

```
indexing
    description: "Access to command-line arguments. This class
      may be used as ancestor by classes needing its facilities."
class interface
    ARGUMENTS
feature -- Access
    argument (i: INTEGER): STRING
            -- i-th argument of command that started system execution
            -- (the command name if i = 0)
        require
            index_large_enough: i >= 0;
            index_small_enough: i <= argument_count
    command_name: STRING
            -- Name of command that started system execution
        ensure
            definition: Result = argument (0)
feature -- Measurement
    argument_count: INTEGER
            -- Number of arguments given to command that started
            -- system execution (command name does not count)
        ensure
            Result >= 0
end
```

5.18 Class *EXCEPTIONS*

indexing

description: *"Facilities for adapting the exception handling mechanism. This class may be used as ancestor by classes needing its facilities."*

class interface

EXCEPTIONS

feature -- Access

developer_exception_name: *STRING*
-- Name of last developer-raised exception
require
applicable: *is_developer_exception*

feature -- Access

Check_instruction: *INTEGER*
-- Exception code for violated check

Class_invariant: *INTEGER*
-- Exception code for violated class invariant

Incorrect_inspect_value: *INTEGER*
-- Exception code for inspect value which is not one
-- of the inspect constants, if there is no Else_part

Loop_invariant: *INTEGER*
-- Exception code for violated loop invariant

Loop_variant: *INTEGER*
-- Exception code for non-decreased loop variant

No_more_memory: *INTEGER*
-- Exception code for failed memory allocation

Postcondition: *INTEGER*
-- Exception code for violated postcondition

Precondition: *INTEGER*
-- Exception code for violated precondition

Routine_failure: *INTEGER*
-- Exception code for failed routine

Void_attached_to_expanded: *INTEGER*
-- Exception code for attachment of void value
-- to expanded entity

Void_call_target: *INTEGER*
-- Exception code for violated check

feature -- Status report

assertion_violation: *BOOLEAN*
-- Is last exception originally due to a violated
-- assertion or non–decreasing variant?

exception: *INTEGER*
-- Code of last exception that occurred

is_developer_exception: *BOOLEAN*
-- Is the last exception originally due to
-- a developer exception?

is_signal: *BOOLEAN*
-- Is last exception originally due to an external
-- event (operating system signal)?

feature -- Basic operations

die (*code*: *INTEGER*)
-- Terminate execution with exit status *code*,
-- without triggering an exception.

raise (*name*: *STRING*)
-- Raise a developer exception of name *name*.

end

5.17 Class *MEMORY*

indexing

description: *"Facilities for tuning up the garbage collection mechanism. This class may be used as ancestor by classes needing its facilities."*

class interface

MEMORY

feature -- Status report

collecting: *BOOLEAN*
-- Is garbage collection enabled?

feature -- Status setting

collection_off
-- Disable garbage collection.

collection_on
-- Enable garbage collection.

feature -- Removal

dispose
-- Action to be executed just before garbage collection
-- reclaims an object.
-- Default version does nothing; redefine in descendants
-- to perform specific dispose actions. Those actions
-- should only take care of freeing external resources;
-- they should not perform remote calls on other objects
-- since these may also be dead and reclaimed.

full_collect
-- Force a full collection cycle if garbage
-- collection is enabled; do nothing otherwise.

end

5.16 Class *STORABLE*

indexing

description: *"Objects that may be stored and retrieved along with all their dependents. This class may be used as ancestor by classes needing its facilities."*

class interface

STORABLE

feature -- Access

retrieved (*file*: *FILE*): *STORABLE*
-- Retrieved object structure, from external
-- representation previously stored in *file*.
-- To access resulting object under correct type,
-- use assignment attempt.
-- Will raise an exception (code *Retrieve_exception*)
-- if file content is not a *STORABLE* structure.

require

file_not_void: *file /= Void;*

file_exists: *file*.*exists;*

file_is_open_read: *file*.*is_open_read*

file_not_plain_text: **not** *file*.*is_plain_text*

ensure

result_exists: *Result /= Void*

feature -- Element change

basic_store (*file*: *FILE*)
-- Produce on *file* an external representation of the
-- entire object structure reachable from current object.
-- Retrievable within current system only.

require

file_not_void: *file /= Void;*

file_exists: *file*.*exists;*

file_is_open_write: *file*.*is_open_write;*

file_not_plain_text: **not** *file*.*is_plain_text*

general_store (*file*: *FILE*)
-- Produce on *file* an external representation of the
-- entire object structure reachable from current object.
-- Retrievable from other systems for same platform
-- (machine architecture).

require

file_not_void: *file /= Void;*

file_exists: *file*.*exists;*

file_is_open_write: *file*.*is_open_write;*

file_not_plain_text: **not** *file*.*is_plain_text*

independent_store (*file*: *FILE*)
-- Produce on *file* an external representation of the
-- entire object structure reachable from current object.
-- Retrievable from other systems for the same or other
-- platforms (machine architectures).

require

file_not_void: *file /= Void;*

file_exists: *file*.*exists;*

file_is_open_write: *file*.*is_open_write;*

file_not_plain_text: **not** *file*.*is_plain_text*

end

```
feature -- Input

    read_character
            -- Read a new character.
            -- Make result available in last_character.
        require
            readable: is_readable

    read_double
            -- Read the ASCII representation of a new double
            -- from file. Make result available in last_double.
        require
            readable: is_readable

    read_integer
            -- Read the ASCII representation of a new integer
            -- from file. Make result available in last_integer.
        require
            readable: is_readable

    read_line
            -- Read a string until new line or end of file.
            -- Make result available in laststring.
            -- New line will be consumed but not part of last_string.
        require
            readable: is_readable

    read_real
            -- Read the ASCII representation of a new real
            -- from file. Make result available in last_real.
        require
            readable: is_readable

    read_stream (nb_char: INTEGER)
            -- Read a string of at most nb_char bound characters
            -- or until end of file.
            -- Make result available in last_string.
        require
            readable: is_readable

    read_word
            -- Read a new word from standard input.
            -- Make result available in last_string.

feature -- Output

    put_boolean (b: BOOLEAN)
            -- Write ASCII value of b at current position.
        require
            extendible: extendible

    put_character (c: CHARACTER)
            -- Write c at current position.
        require
            extendible: extendible

    put_double (d: DOUBLE)
            -- Write ASCII value of d at current position.
        require
            extendible: extendible

    put_integer (i: INTEGER)
            -- Write ASCII value of i at current position.
        require
            extendible: extendible

    put_real (r: REAL)
            -- Write ASCII value of r at current position.
        require
            extendible: extendible

    put_string (s: STRING)
            -- Write s at current position.
        require
            extendible: extendible

invariant

    name_exists: name /= Void;

    name_not_empty: not name.empty;

    writable_if_extendible: extendible implies is_writable

end
```

```
exists: BOOLEAN
        -- Does physical file exist?

is_closed: BOOLEAN
        -- Is file closed?

is_open_read: BOOLEAN
        -- Is file open for reading?

is_open_write: BOOLEAN
        -- Is file open for writing?

is_plain_text: BOOLEAN
        -- Is file reserved for text (character sequences)?

is_readable: BOOLEAN
        -- Is file readable?
    require
        handle_exists: exists

is_writable: BOOLEAN
        -- Is file writable?
    require
        handle_exists: exists

last_character: CHARACTER
        -- Last character read by read_character

last_double: DOUBLE
        -- Last double read by read_double

last_integer: INTEGER
        -- Last integer read by read_integer

last_real: REAL
        -- Last real read by read_real

last_string: STRING
        -- Last string read by read_line,
        -- read_stream, or read_word

feature -- Status setting

close
        -- Close file.
    require
        medium_is_open: not is_closed
    ensure
        is_closed: is_closed

open_read
        -- Open file in read–only mode.
    require
        is_closed: is_closed
    ensure
        exists: exists;
        open_read: is_open_read

open_read_append
        -- Open file in read and write–at-end mode;
        -- create it if it does not exist.
    require
        is_closed: is_closed
    ensure
        exists: exists;
        open_read: is_open_read;
        open_append: is_open_append

open_read_write
        -- Open file in read and write mode.
    require
        is_closed: is_closed
    ensure
        exists: exists;
        open_read: is_open_read;
        open_write: is_open_write

open_write
        -- Open file in write–only mode;
        -- create it if it does not exist.
    ensure
        exists: exists;
        open_write: is_open_write

feature -- Cursor movement

to_next_line
        -- Move to next input line.
    require
        readable: is_readable

feature -- Element change

change_name (new_name: STRING)
        -- Change file name to new_name
    require
        not_new_name_void: new_name /= Void;
        file_exists: exists
    ensure
        name_changed: name.is_equal (new_name)

feature -- Removal

delete
        -- Remove link with physical file; delete physical
        -- file if no more link.
    require
        exists: exists

dispose
        -- Ensure this medium is closed when garbage-collected.
```

5.15 Class *FILE*

indexing

description: *"Files viewed as persistent sequences of characters"*

class interface

FILE

creation

make (*fn*: *STRING*)
-- Create file object with *fn* as file name.
require
string_exists: *fn /= Void;*
string_not_empty: ***not*** *fn*.*empty*
ensure
file_named: *name*.*is_equal* (*fn*);
file_closed: *is_closed*

make_create_read_write (*fn*: *STRING*)
-- Create file object with *fn* as file name
-- and open file for both reading and writing;
-- create it if it does not exist.
require
string_exists: *fn /= Void;*
string_not_empty: ***not*** *fn*.*empty*
ensure
exists: *exists;*
open_read: *is_open_read;*
open_write: *is_open_write*

make_open_append (*fn*: *STRING*)
-- Create file object with *fn* as file name
-- and open file in append–only mode.
require
string_exists: *fn /= Void;*
string_not_empty: ***not*** *fn*.*empty*
ensure
exists: *exists;*
open_append: *is_open_append*

make_open_read (*fn*: *STRING*)
-- Create file object with *fn* as file name
-- and open file in read mode.
require
string_exists: *fn /= Void;*
string_not_empty: ***not*** *fn*.*empty*
ensure
exists: *exists;*
open_read: *is_open_read*

make_open_read_write (*fn*: *STRING*)
-- Create file object with *fn* as file name
-- and open file for both reading and writing.
require
string_exists: *fn /= Void;*
string_not_empty: ***not*** *fn*.*empty*
ensure
exists: *exists;*
open_read: *is_open_read;*
open_write: *is_open_write*

make_open_write (*fn*: *STRING*)
-- Create file object with *fn* as file name
-- and open file for writing;
-- create it if it does not exist.
require
string_exists: *fn /= Void;*
string_not_empty: ***not*** *fn*.*empty*
ensure
exists: *exists;*
open_write: *is_open_write*

feature -- Access

name: *STRING*
-- File name

feature -- Measurement

count: *INTEGER*
-- Size in bytes (0 if no associated physical file)

feature -- Status report

empty: *BOOLEAN*
-- Is structure empty?

end_of_file: *BOOLEAN*
-- Has an EOF been detected?
require
opened: ***not*** *is_closed*

5.14 Class *STD_FILES*

indexing

description: *"Commonly used input and output mechanisms. This class may be used as either ancestor or supplier by classes needing its facilities."*

class interface

STD_FILES

feature -- Access

default_output: *FILE*
-- Default output.

error: *FILE*
-- Standard error file

input: *FILE*
-- Standard input file

output: *FILE*
-- Standard output file

standard_default: *FILE*
-- Return the *default_output* or *output*
-- if *default_output* is Void.

feature -- Status report

last_character: *CHARACTER*
-- Last character read by *read_character*

last_double: *DOUBLE*
-- Last double read by *read_double*

last_integer: *INTEGER*
-- Last integer read by *read_integer*

last_real: *REAL*
-- Last real read by *read_real*

last_string: *STRING*
-- Last string read by *read_line*,
-- *read_stream*, or *read_word*

feature -- Element change

put_boolean (*b*: *BOOLEAN*)
-- Write *b* at end of default output.

put_character (*c*: *CHARACTER*)
-- Write *c* at end of default output.

put_double (*d*: *DOUBLE*)
-- Write *d* at end of default output.

put_integer (*i*: *INTEGER*)
-- Write *i* at end of default output.

put_new_line
-- Write line feed at end of default output.

put_real (*r*: *REAL*)
-- Write *r* at end of default output.

put_string (*s*: *STRING*)
-- Write *s* at end of default output.

require

s /= Void

set_error_default
-- Use standard error as default output.

set_output_default
-- Use standard output as default output.

feature -- Input

read_character
-- Read a new character from standard input.
-- Make result available in *last_character*.

read_double
-- Read a new double from standard input.
-- Make result available in *last_double*.

read_integer
-- Read a new integer from standard input.
-- Make result available in *last_integer*.

read_line
-- Read a line from standard input.
-- Make result available in *last_string*.
-- New line will be consumed but not part of *last_string*.

read_real
-- Read a new real from standard input.
-- Make result available in *last_real*.

read_stream (*nb_char*: *INTEGER*)
-- Read a string of at most *nb_char* bound characters
-- from standard input.
-- Make result available in *last_string*.

to_next_line
-- Move to next input line on standard input.

end

```
to_upper
        -- Convert to upper case.

feature -- Duplication

    copy (other: like Current)
            -- Reinitialize by copying the characters of other.
            -- (This is also used by clone.)
            -- (From GENERAL.)
        ensure
            new_result_count: count = other.count
            -- same_characters: For every i in 1..count,
            --      item (i) = other.item (i)

    substring (n1, n2: INTEGER): like Current
            -- Copy of substring containing all characters at indices
            -- between n1 and n2
        require
            meaningful_origin: 1 <= n1;
            meaningful_interval: n1 <= n2;
            meaningful_end: n2 <= count
        ensure
            new_result_count: Result.count = n2 – n1 + 1
            -- original_characters: For every i in 1..n2–n1,
            --      Result.item (i) = item (n1+i–1)

feature -- Output

    out: STRING
            -- Printable representation
            -- (From GENERAL.)
        ensure
            result_not_void: Result /= Void

invariant

    irreflexive_comparison: not (Current < Current);

    empty_definition: empty = (count = 0);

    non_negative_count: count >= 0

end
```

fill (*c: CHARACTER*)
-- Replace every character with *c*.
ensure
-- *allblank*: For every *i* in 1..*count*, *item* (*i*) = *Blank*

head (*n*: *INTEGER*)
-- Remove all characters except for the first *n*;
-- do nothing if *n* >= *count*.
require
non_negative_argument: *n >= 0*
ensure
new_count: *count* = *n*.*min* (***old*** *count*)
-- *first_kept*: For every *i* in 1..*n*, *item* (*i*) = ***old*** *item* (*i*)

insert (*s*: ***like*** *Current; i*: *INTEGER*)
-- Add *s* to the left of position *i*.
require
string_exists: *s /= Void;*
index_small_enough: *i <= count;*
index_large_enough: *i > 0*
ensure
new_count: *count* = ***old*** *count + s*.*count*

insert_character (*c*: *CHARACTER; i*: *INTEGER*)
-- Add *c* to the left of position *i*.
ensure
new_count: *count* = ***old*** *count + 1*

left_adjust
-- Remove leading white space.
ensure
new_count: (*count /= 0*) ***implies*** (*item* (*1*) /= ' ')

put (*c*: *CHARACTER; i*: *INTEGER*)
-- Replace character at position *i* by *c*.
require
good_key: *valid_index* (*i*)
ensure
insertion_done: *item* (*i*) = *c*

put_substring (*s*: ***like*** *Current; start_pos*, *end_pos*: *INTEGER*)
-- Copy the characters of *s* to positions
-- *start_pos* .. *end_pos*.
require
string_exists: *s /= Void;*
index_small_enough: *end_pos <= count;*
order_respected: *start_pos <= end_pos;*
index_large_enough: *start_pos > 0*
ensure
new_count: *count* = ***old*** *count + s*.*count – end_pos + start_pos – 1*

right_adjust
-- Remove trailing white space.
ensure
new_count: (*count /= 0*) ***implies*** (*item* (*count*) /= ' ')

tail (*n*: *INTEGER*)
-- Remove all characters except for the last *n*;
-- do nothing if *n* >= *count*.
require
non_negative_argument: *n >= 0*
ensure
new_count: *count* = *n*.*min* (***old*** *count*)

feature -- Removal

remove (*i*: *INTEGER*)
-- Remove *i*–th character.
require
index_small_enough: *i <= count;*
index_large_enough: *i > 0*
ensure
new_count: *count* = ***old*** *count – 1*

wipe_out
-- Remove all characters.
ensure
empty_string: *count = 0;*
wiped_out: *empty*

feature -- Resizing

resize (*newsize*: *INTEGER*)
-- Rearrange string so that it can accommodate
-- at least *newsize* characters.
-- Do not lose any previously entered character.
require
new_size_non_negative: *newsize >= 0*

feature -- Conversion

to_boolean: *BOOLEAN*
-- Boolean value;
-- "true" yields *true*, "false" yields *false*
-- (case–insensitive)

to_double: *DOUBLE*
-- "Double" value; for example, when applied to "123.0",
-- will yield 123.0 (double)

to_integer: *INTEGER*
-- Integer value;
-- for example, when applied to "123", will yield 123

to_lower
-- Convert to lower case.

to_real: *REAL*
-- Real value;
-- for example, when applied to "123.0", will yield 123.0

infix *"<"* (*other*: *STRING*): *BOOLEAN*
-- Is string lexicographically lower than *other*?
-- (False if *other* is void)
-- (From *COMPARABLE*.)
require
other_exists: *other* /= *Void*
ensure
asymmetric: *Result* ***implies not*** (*other* < *Current*)

infix *"<="* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object less than or equal to *other*?
-- (From *COMPARABLE*.)
require
other_exists: *other* /= *Void*
ensure
definition: *Result* = (*Current* < *other*) ***or*** *is_equal* (*other*);

infix *">="* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object greater than or equal to *other*?
-- (From *COMPARABLE*.)
require
other_exists: *other* /= *Void*
ensure
definition: *Result* = (*other* <= *Current*)

infix *">"* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object greater than *other*?
-- (From *COMPARABLE*.)
require
other_exists: *other* /= *Void*
ensure
definition: *Result* = (*other* < *Current*)

max (*other*: ***like*** *Current*): ***like*** *Current*)
-- The greater of current object and *other*
-- (From *COMPARABLE*.)
require
other_exists: *other* /= *Void*
ensure
current_if_not_smaller: (*Current* >= *other*) ***implies*** (*Result* = *Current*)
other_if_smaller: (*Current* < *other*) ***implies*** (*Result* = *other*)

min (*other*: ***like*** *Current*): ***like*** *Current*)
-- The smaller of current object and *other*
-- (From *COMPARABLE*.)
require
other_exists: *other* /= *Void*
ensure
current_if_not_greater: (*Current* <= *other*) ***implies*** (*Result* = *Current*)
other_if_greater: (*Current* > *other*) ***implies*** (*Result* = *other*)

three_way_comparison (*other*: ***like*** *Current*): *INTEGER*)
-- If current object equal to *other*, 0; if smaller,
-- –1; if greater, 1.
-- (From *COMPARABLE*.)
require
other_exists: *other* /= *Void*
ensure
equal_zero: (*Result* = *0*) = *is_equal* (*other*);
smaller: (*Result* = *–1*) = *Current* < *other*;
greater_positive: (*Result* = *1*) = *Current* > *other*

feature -- Status report

empty: *BOOLEAN*
-- Is string empty?

valid_index (*i*: *INTEGER*): *BOOLEAN*
-- Is *i* within the bounds of the string?

feature -- Element change

append_boolean (*b*: *BOOLEAN*)
-- Append the string representation of *b* at end.

append_character (*c*: *CHARACTER*)
-- Append *c* at end.
ensure
item_inserted: *item* (*count*) = *c*
one_more_occurrence: *occurrences* (*c*) = ***old*** (*occurrences* (*c*)) + *1*
item_inserted: *has* (*c*)

append_double (*d*: *DOUBLE*)
-- Append the string representation of *d* at end.

append_integer (*i*: *INTEGER*)
-- Append the string representation of *i* at end.

append_real (*r*: *REAL*)
-- Append the string representation of *r* at end.

append_string (*s*: *STRING*)
-- Append a copy of *s*, if not void, at end.
ensure
new_count: *count* = ***old*** *count* + *s*.*count*
-- *appended*: For every *i* in 1..*s*.*count*,
-- *item* (***old*** *count* + *i*) = *s*.*item* (*i*)

5.13 Class *STRING*

indexing

description: *"Sequences of characters, accessible through integer indices in a contiguous range."*

class interface

STRING

creation

frozen *make* (*n*: *INTEGER*)
-- Allocate space for at least *n* characters.
require
non_negative_size: *n >= 0*
ensure
empty_string: *count = 0*

make_from_string (*s*: *STRING*)
-- Initialize from the characters of *s*.
-- (Useful in proper descendants of class *STRING,*
-- to initialize a string-like object from a manifest string.)
require
string_exists: s/= Void

feature -- Initialization

from_c (*c_string*: *POINTER*)
-- Reset contents of string from contents of *c_string*,
-- a string created by some external C function.
require
C_string_exists: c_string /= Void

frozen *remake* (*n*: *INTEGER*)
-- Allocate space for at least *n* characters.
require
non_negative_size: *n >= 0*
ensure
empty_string: *count = 0*

make_from_string (*s*: *STRING*)
-- Initialize from the characters of *s*.
-- (Useful in proper descendants of class *STRING,*
-- to initialize a string-like object from a manifest string.)
require
string_exists: s/= Void

feature -- Access

hash_code: *INTEGER*
-- Hash code value
-- (From *HASHABLE*.)
ensure
good_hash_value: *Result >= 0*

index_of (*c*: *CHARACTER; start*: *INTEGER*): *INTEGER*
-- Position of first occurrence of *c* at or after *start*;
-- 0 if none.
require
start_large_enough: *start >= 1*;
start_small_enough: *start <= count*
ensure
non_negative_result: *Result >= 0*;
at_this_position: *Result > 0* ***implies*** *item* (*Result*) = *c*;
-- *none_before*: For every *i* in *start..Result*, *item* (*i*) /= *c*
-- *zero_iff_absent*:
-- (*Result* = *0*) = For every *i* in 1..*count*, *item* (*i*) /= *c*

item (*i*: *INTEGER*): *CHARACTER*
-- Character at position *i*
require
good_key: *valid_index* (*i*)

substring_index (*other*: *STRING; start*: *INTEGER*): *INTEGER*
-- Position of first occurrence of *other* at or after *start*;
-- 0 if none.

infix *"@"* (*i*: *INTEGER*): *CHARACTER*
-- Character at position *i*
require
good_key: *valid_index* (*i*)

feature -- Measurement

count: *INTEGER*
-- Actual number of characters making up the string

occurrences (*c*: *CHARACTER*): *INTEGER*
-- Number of times *c* appears in the string
ensure
non_negative_occurrences: *Result >= 0*

feature -- Comparison

is_equal (*other*: ***like*** *Current*): *BOOLEAN*
-- Is string made of same character sequence as *other*?
-- (Redefined from *GENERAL*.)
require
other_exists: *other /= Void*

```
frozen put (v: like item; i: INTEGER)
        -- Replace i–th entry, if in index interval, by v.
    require
        good_key: valid_index (i)
    ensure
        inserted: item (i) = v
feature -- Resizing
    resize (minindex, maxindex: INTEGER)
        -- Rearrange array so that it can accommodate
        -- indices down to minindex and up to maxindex.
        -- Do not lose any previously entered item.
    require
        good_indices: minindex <= maxindex
    ensure
        no_low_lost: lower = minindex.min (old lower);
        no_high_lost: upper = maxindex.max (old upper)
feature -- Conversion
    to_c: POINTER
        -- Address of actual sequence of values,
        -- for passing to external (non–Eiffel) routines.
feature -- Duplication
    copy (other: like Current)
        -- Reinitialize by copying all the items of other.
        -- (This is also used by clone.)
        -- (From GENERAL.)
invariant
    consistent_size: count = upper – lower + 1;
    non_negative_count: count >= 0
end
```

5.12 Class *ARRAY*

indexing

description: *"Sequences of values, all of the same type or of a conforming one, accessible through integer indices in a contiguous interval"*

class interface

ARRAY [*G*]

creation

make (*minindex*, *maxindex*: *INTEGER*)
-- Allocate array; set index interval to
-- *minindex* .. *maxindex*; set all values to default.
-- (Make array empty if *minindex* > *maxindex*.)

ensure

no_count: (*minindex* > *maxindex*) ***implies*** (*count* = *0*);

count_constraint: (*minindex* <= *maxindex*) ***implies*** (*count* = *maxindex* – *minindex* + *1*)

make_from_array (*a*: *ARRAY* [*G*])
-- Initialize from the items of *a*.
-- (Useful in proper descendants of class *ARRAY*,
-- to initialize an array-like object from a manifest array.)

feature -- Initialization

make (*minindex*, *maxindex*: *INTEGER*)
-- Set index interval to *minindex* .. *maxindex*;
-- reallocate if necessary; set all values to default.
-- (Make array empty if *minindex* > *maxindex*.)

ensure

no_count: (*minindex* > *maxindex*) ***implies*** (*count* = *0*);

count_constraint: (*minindex* <= *maxindex*) ***implies*** (*count* = *maxindex* – *minindex* + *1*)

make_from_array (*a*: *ARRAY* [*G*])
-- Initialize from the items of *a*; reallocate if
-- necessary. (Useful in proper descendants of
-- class *ARRAY*, to initialize an array-like object
-- from a manifest array.)

feature -- Access

entry (*i*: *INTEGER*): *G*
-- Entry at index *i*, if in index interval
-- (Redefinable synonym for *item* and ***infix*** "@".)

require

good_key: *valid_index* (*i*)

frozen *item* (*i*: *INTEGER*): *G*
-- Entry at index *i*, if in index interval

require

good_key: *valid_index* (*i*)

frozen infix *"@"* (*i*: *INTEGER*): *G*
-- Entry at index *i*, if in index interval

require

good_key: *valid_index* (*i*)

feature -- Measurement

count: *INTEGER*
-- Number of available indices

lower: *INTEGER*
-- Minimum index

upper: *INTEGER*
-- Maximum index

feature -- Comparison

is_equal (*other*: ***like*** *Current*): *BOOLEAN*
-- Is array made of the same items as *other*?
-- (Redefined from *GENERAL*.)

feature -- Status report

valid_index (*i*: *INTEGER*): *BOOLEAN*
-- Is *i* within the bounds of the array?

feature -- Element change

enter (*v*: *G; i*: *INTEGER*)
-- Replace *i*–th entry, if in index interval, by *v*.
-- (Redefinable synonym for *put*.)

require

good_key: *valid_index* (*i*)

ensure

inserted: *item* (*i*) = *v*

force (*v*: ***like*** *item; i*: *INTEGER*)
-- Assign item *v* to *i*–th entry.
-- Always applicable: resize the array if *i* falls out of
-- currently defined bounds; preserve existing items.

ensure

inserted: *item* (*i*) = *v;*

higher_count: *count* >= ***old*** *count*

5.11 Class *POINTER*

indexing

description: *"References to objects meant to be exchanged with non-Eiffel software."*

deferred class interface

POINTER

feature -- Access

hash_code: *INTEGER*
-- Hash code value
-- (From *HASHABLE*.)
ensure
good_hash_value: *Result >= 0*

feature -- Output

out: *STRING*
-- Printable representation of pointer value
-- (From *GENERAL*.)

end

```
prefix "–": like Current
        -- Unary minus
        -- (From NUMERIC.)
    ensure
        result_exists: Result /= Void
feature -- Output
    out: STRING
        -- Printable representation of double value
        -- (From GENERAL.)
invariant
    irreflexive_comparison: not (Current < Current);
    neutral_addition: equal (Current + zero, Current);
    self_subtraction: equal (Current – Current, zero);
    neutral_multiplication: equal (Current * one, Current);
    self_division: divisible (Current) implies equal (Current /
      Current, one)
    sign_times_abs: equal (sign*abs, Current)
end
```

```
feature -- Status report
    divisible (other: like Current): BOOLEAN
            -- May current object be divided by other?
            -- (From NUMERIC.)
        require
            other_exists: other /= Void
        ensure
            not_exact_zero: Result implies (other /= 0.0)
    exponentiable (other: NUMERIC): BOOLEAN
            -- May current object be elevated to the power other?
            -- (From NUMERIC.)
        require
            other_exists: other /= Void
        ensure
            safe_values: (other.conforms_to (0) or
              (other.conforms_to (Current) and (Current >= 0.0)))
                implies Result
feature -- Conversion
    ceiling: INTEGER
            -- Smallest integral value no smaller than current object
        ensure
            result_no_smaller: Result >= Current;
            close_enough: Result – Current < one
    floor: INTEGER
            -- Greatest integral value no greater than current object
        ensure
            result_no_greater: Result <= Current;
            close_enough: Current – Result < one
    rounded: INTEGER
            -- Rounded integral value
        ensure
            definition: Result = sign * ((abs + 0.5).floor)
    truncated_to_integer: INTEGER
            -- Integer part (same sign, largest absolute
            -- value no greater than current object's)
    truncated_to_real: REAL
            -- Real part (same sign, largest absolute
            -- value no greater than current object's)
feature -- Basic operations
    abs: like Current
            -- Absolute value
        ensure
            non_negative: Result >= 0;
            same_absolute_value: (Result = Current) or (Result = –
              Current)
    infix "*" (other: like Current): like Current
            -- Product by other
            -- (From NUMERIC.)
        require
            other_exists: other /= Void
        ensure
            result_exists: Result /= Void
    infix "+" (other: like Current): like Current
            -- Sum with other
            -- (From NUMERIC.)
        require
            other_exists: other /= Void
        ensure
            result_exists: Result /= Void;
            commutative: equal (Result, other + Current)
    infix "–" (other: like Current): like Current
            -- Result of subtracting other
            -- (From NUMERIC.)
        require
            other_exists: other /= Void
        ensure
            result_exists: Result /= Void
    infix "/" (other: like Current): like Current
            -- Division by other
            -- (From NUMERIC.)
        require
            other_exists: other /= Void;
            good_divisor: divisible (other)
        ensure
            result_exists: Result /= Void
    infix "^" (other: like Current): like Current
            -- Current double to the power other
            -- (From NUMERIC.)
        require
            other_exists: other /= Void;
            good_exponent: exponentiable (other)
        ensure
            result_exists: Result /= Void
    prefix "+": like Current
            -- Unary plus
            -- (From NUMERIC.)
        ensure
            result_exists: Result /= Void
```

5.10 Class *DOUBLE*

indexing

description: *"Real values, double precision"*

expanded class interface

DOUBLE

feature -- Access

hash_code: *INTEGER*
-- Hash code value
-- (From *HASHABLE*.)
ensure
good_hash_value: *Result >= 0*

one: ***like*** *Current*
-- Neutral element for "*" and "/"
-- (From *NUMERIC*.)
ensure
Result_exists: *Result /= Void*;
value: *Result = 1.0*

sign: *INTEGER*
-- Sign value (0, –1 or 1)
ensure
three_way: *Result = three_way_comparison* (*zero*)

zero: ***like*** *Current*
-- Neutral element for "+" and "–"
-- (From *NUMERIC*.)
ensure
Result_exists: *Result /= Void*;
value: *Result = 0.0*

feature -- Comparison

infix *"<"* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is *other* greater than current double?
-- (From *COMPARABLE*.)
require
other_exists: *other /= Void*
ensure
asymmetric: *Result* **implies not** (*other < Current*)

infix *"<="* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object less than or equal to *other*?
-- (From *COMPARABLE*.)
require
other_exists: *other /= Void*
ensure
definition: *Result* = (*Current < other*) ***or*** *is_equal* (*other*);

infix *">="* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object greater than or equal to *other*?
-- (From *COMPARABLE*.)
require
other_exists: *other /= Void*
ensure
definition: *Result* = (*other <= Current*)

infix *">"* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object greater than *other*?
-- (From *COMPARABLE*.)
require
other_exists: *other /= Void*
ensure
definition: *Result* = (*other < Current*)

max (*other*: ***like*** *Current*): ***like*** *Current*
-- The greater of current object and *other*
-- (From *COMPARABLE*.)
require
other_exists: *other /= Void*
ensure
current_if_not_smaller: (*Current >= other*) ***implies*** (*Result = Current*)
other_if_smaller: (*Current < other*) ***implies*** (*Result = other*)

min (*other*: ***like*** *Current*): ***like*** *Current*
-- The smaller of current object and *other*
-- (From *COMPARABLE*.)
require
other_exists: *other /= Void*
ensure
current_if_not_greater: (*Current <= other*) ***implies*** (*Result = Current*)
other_if_greater: (*Current > other*) ***implies*** (*Result = other*)

three_way_comparison (*other*: ***like*** *Current*): *INTEGER*
-- If current object equal to *other*, 0; if smaller,
-- –1; if greater, 1.
require
other_exists: *other /= Void*
-- (From *COMPARABLE*.)
ensure
equal_zero: (*Result = 0*) = *is_equal* (*other*);
smaller: (*Result = –1*) = *Current < other*;
greater_positive: (*Result = 1*) = *Current > other*

```
prefix "–": like Current
        -- Unary minus
        -- (From NUMERIC.)
    ensure
        result_exists: Result /= Void

feature -- Output

    out: STRING
        -- Printable representation of real value
        -- (From GENERAL.)

invariant

    irreflexive_comparison: not (Current < Current);

    neutral_addition: equal (Current + zero, Current);

    self_subtraction: equal (Current – Current, zero);

    neutral_multiplication: equal (Current * one, Current);

    self_division: divisible (Current) implies equal (Current /
      Current, one)

    sign_times_abs: equal (sign*abs, Current)

end
```

```
feature -- Status report

    divisible (other: like Current): BOOLEAN
            -- May current object be divided by other?
            -- (From NUMERIC.)
        require
            other_exists: other /= Void
        ensure
            not_exact_zero: Result implies (other /= 0.0)

    exponentiable (other: NUMERIC): BOOLEAN
            -- May current object be elevated to the power other?
            -- (From NUMERIC.)
        require
            other_exists: other /= Void
        ensure
            safe_values: (other.conforms_to (0) or
              (other.conforms_to (Current) and (Current >= 0.0)))
                implies Result

feature -- Conversion

    ceiling: INTEGER
            -- Smallest integral value no smaller than current object
        ensure
            result_no_smaller: Result >= Current;
            close_enough: Result – Current < one

    floor: INTEGER
            -- Greatest integral value no greater than current object
        ensure
            result_no_greater: Result <= Current;
            close_enough: Current – Result < one

    rounded: INTEGER
            -- Rounded integral value
        ensure
            definition: Result = sign * ((abs + 0.5).floor)

    truncated_to_integer: INTEGER
            -- Integer part (same sign, largest absolute
            -- value no greater than current object's)

feature -- Basic operations

    abs: like Current
            -- Absolute value
        ensure
            non_negative: Result >= 0;
            same_absolute_value: (Result = Current) or (Result = –
              Current)

    infix "*" (other: like Current): like Current
            -- Product by other
            -- (From NUMERIC.)
        require
            other_exists: other /= Void
        ensure
            result_exists: Result /= Void

    infix "+" (other: like Current): like Current
            -- Sum with other
            -- (From NUMERIC.)
        require
            other_exists: other /= Void
        ensure
            result_exists: Result /= Void;
            commutative: equal (Result, other + Current)

    infix "–" (other: like Current): like Current
            -- Result of subtracting other
            -- (From NUMERIC.)
        require
            other_exists: other /= Void
        ensure
            result_exists: Result /= Void

    infix "/" (other: like Current): like Current
            -- Division by other
            -- (From NUMERIC.)
        require
            other_exists: other /= Void;
            good_divisor: divisible (other)
        ensure
            result_exists: Result /= Void

    infix "^" (other: NUMERIC): DOUBLE
            -- Current real to the power other
            -- (From NUMERIC.)
        require
            other_exists: other /= Void;
            good_exponent: exponentiable (other)
        ensure
            result_exists: Result /= Void

    prefix "+": like Current
            -- Unary plus
            -- (From NUMERIC.)
        ensure
            result_exists: Result /= Void
```

5.9 Class *REAL*

indexing

description: *"Real values, single precision"*

expanded class interface

REAL

feature -- Access

hash_code: *INTEGER*
-- Hash code value
-- (From *HASHABLE*.)
ensure
good_hash_value: *Result >= 0*

one: ***like*** *Current*
-- Neutral element for "*" and "/"
-- (From *NUMERIC*.)
ensure
Result_exists: *Result /= Void*;
value: *Result = 1.0*

sign: *INTEGER*
-- Sign value (0, –1 or 1)
ensure
three_way: *Result = three_way_comparison* (*zero*)

zero: ***like*** *Current*
-- Neutral element for "+" and "–"
-- (From *NUMERIC*.)
ensure
Result_exists: *Result /= Void*;
value: *Result = 0.0*

feature -- Comparison

infix *"<"* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is *other* greater than current real?
-- (From *COMPARABLE*.)
require
other_exists: *other /= Void*
ensure
asymmetric: *Result* ***implies not*** (*other < Current*)

infix *"<="* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object less than or equal to *other*?
-- (From *COMPARABLE*.)
require
other_exists: *other /= Void*
ensure
definition: *Result* = (*Current < other*) ***or*** *is_equal* (*other*);

infix *">="* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object greater than or equal to *other*?
-- (From *COMPARABLE*.)
require
other_exists: *other /= Void*
ensure
definition: *Result =* (*other <= Current*)

infix *">"* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object greater than *other*?
-- (From *COMPARABLE*.)
require
other_exists: *other /= Void*
ensure
definition: *Result =* (*other < Current*)

max (*other*: ***like*** *Current*): ***like*** *Current*
-- The greater of current object and *other*
-- (From *COMPARABLE*.)
require
other_exists: *other /= Void*
ensure
current_if_not_smaller: (*Current >= other*) ***implies*** (*Result = Current*)
other_if_smaller: (*Current < other*) ***implies*** (*Result = other*)

min (*other*: ***like*** *Current*): ***like*** *Current*
-- The smaller of current object and *other*
-- (From *COMPARABLE*.)
require
other_exists: *other /= Void*
ensure
current_if_not_greater: (*Current <= other*) ***implies*** (*Result = Current*)
other_if_greater: (*Current > other*) ***implies*** (*Result = other*)

three_way_comparison (*other*: ***like*** *Current*): *INTEGER*
-- If current object equal to *other*, 0; if smaller,
-- –1; if greater, 1.
-- (From *COMPARABLE*.)
require
other_exists: *other /= Void*
ensure
equal_zero: (*Result = 0*) = *is_equal* (*other*);
smaller: (*Result = –1*) = *Current < other*;
greater_positive: (*Result = 1*) = *Current > other*

feature -- Output

out: *STRING*
-- Printable representation of current object
-- (From *GENERAL*.)

invariant

irreflexive_comparison: ***not*** (*Current* < *Current*);

neutral_addition: *equal* (*Current* + *zero*, *Current*);

self_subtraction: *equal* (*Current* – *Current*, *zero*);

neutral_multiplication: *equal* (*Current* * *one*, *Current*);

self_division: *divisible* (*Current*) ***implies*** *equal* (*Current* / *Current*, *one*)

sign_times_abs: *equal* (*sign**abs*, *Current)*

end

three_way_comparison (*other*: ***like*** *Current*): *INTEGER*
-- If current object equal to *other*, 0; if smaller,
-- –1; if greater, 1.
-- (From *COMPARABLE*.)
require
other_exists: *other* /= *Void*
ensure
equal_zero: (*Result* = *0*) = *is_equal* (*other*);
smaller: (*Result* = *1*) = *Current* < *other*;
greater_positive: (*Result* = –*1*) = *Current* > *other*

feature -- Status report

divisible (*other*: ***like*** *Current*): *BOOLEAN*
-- May current object be divided by *other*?
-- (From *NUMERIC*.)
require
other_exists: *other* /= *Void*
ensure
value: *Result* = (*other* /= *0*)

exponentiable (*other*: *NUMERIC*): *BOOLEAN*
-- May current object be elevated to the power *other*?
-- (From *NUMERIC*.)
require
other_exists: *other* /= *Void*
ensure
safe_values: (*other*.*conforms_to* (*Current*) ***or***
(*other*.*conforms_to* (*0*.*0*) ***and*** (*Current* >= *0*)))
implies *Result*

feature -- Basic operations

abs: ***like*** *Current*
-- Absolute value
ensure
non_negative: *Result* >= *0*;
same_absolute_value: (*Result* = *Current*) ***or*** (*Result* = –*Current*)

infix *"*"* (*other*: ***like*** *Current*): ***like*** *Current*
-- Product by *other*
-- (From *NUMERIC*.)
require
other_exists: *other* /= *Void*

infix *"+"* (*other*: ***like*** *Current*): ***like*** *Current*
-- Sum with *other*
-- (From *NUMERIC*.)
require
other_exists: *other* /= *Void*
ensure
result_exists: *Result* /= *Void*;
commutative: *equal* (*Result*, *other* + *Current*)

infix *"–"* (*other*: ***like*** *Current*): ***like*** *Current*
-- Result of subtracting *other*
-- (From *NUMERIC*.)
require
other_exists: *other* /= *Void*
ensure
result_exists: *Result* /= *Void*

infix *"/"* (*other*: ***like*** *Current*): *DOUBLE*
-- Division by *other*
require
other_exists: *other* /= *Void*;
good_divisor: *divisible* (*other*)
ensure
result_exists: *Result* /= *Void*

infix *"//"* (*other*: ***like*** *Current*): ***like*** *Current*
-- Integer division of Current by *other*
-- (From *"/"* in *NUMERIC*.)
require
other_exists: *other* /= *Void*;
good_divisor: *divisible* (*other*)
ensure
result_exists: *divisible* (*other*)

infix *"\\"* (*other*: ***like*** *Current*): ***like*** *Current*
-- Remainder of the integer division of Current by *other*
require
other_exists: *other* /= *Void*;
good_divisor: *divisible* (*other*)
ensure
result_exists: *Result* /= *Void*

infix *"^"* (*other*: *NUMERIC*): *DOUBLE*
-- Integer power of Current by *other*
-- (From *NUMERIC*.)
require
other_exists: *other* /= *Void*;
good_exponent: *exponentiable* (*other*)
ensure
result_exists: *Result* /= *Void*

prefix *"+"*: ***like*** *Current*
-- Unary plus
-- (From *NUMERIC*.)
ensure
result_exists: *Result* /= *Void*

prefix *"–"*: ***like*** *Current*
-- Unary minus
-- (From *NUMERIC*.)
ensure
result_exists: *Result* /= *Void*

5.8 Class *INTEGER*

```
indexing

    description: "Integer values"

expanded class interface

    INTEGER

feature -- Access

    hash_code: INTEGER is
            -- Hash code value
            -- (From HASHABLE.)
        ensure
            good_hash_value: Result >= 0

    one: like Current
            -- Neutral element for "*" and "/"
            -- (From NUMERIC.)
        ensure
            Result_exists: Result /= Void;
            value: Result = 1

    sign: INTEGER
            -- Sign value (0, –1 or 1)
        ensure
            three_way: Result = three_way_comparison (zero)

    zero: like Current
            -- Neutral element for "+" and "–"
            -- (From NUMERIC.)
        ensure
            Result_exists: Result /= Void;
            value: Result = 0

feature -- Comparison

    infix "<" (other: like Current): BOOLEAN
            -- Is other greater than current integer?
            -- (From COMPARABLE.)
        require
            other_exists: other /= Void
        ensure
            asymmetric: Result implies not (other < Current)

    infix "<=" (other: like Current): BOOLEAN
            -- Is current object less than or equal to other?
            -- (From COMPARABLE.)
        require
            other_exists: other /= Void
        ensure
            definition: Result = (Current < other) or is_equal
              (other);

    infix ">=" (other: like Current): BOOLEAN
            -- Is current object greater than or equal to other?
            -- (From COMPARABLE.)
        require
            other_exists: other /= Void
        ensure
            definition: Result = (other <= Current)

    infix ">" (other: like Current): BOOLEAN
            -- Is current object greater than other?
            -- (From COMPARABLE.)
        require
            other_exists: other /= Void
        ensure
            definition: Result = (other < Current)

    max (other: like Current): like Current
            -- The greater of current object and other
            -- (From COMPARABLE.)
        require
            other_exists: other /= Void
        ensure
            current_if_not_smaller: (Current >= other) implies
              (Result = Current)
            other_if_smaller: (Current < other) implies (Result =
              other)

    min (other: like Current): like Current
            -- The smaller of current object and other
            -- (From COMPARABLE.)
        require
            other_exists: other /= Void
        ensure
            current_if_not_greater: (Current <= other) implies
              (Result = Current)
            other_if_greater: (Current > other) implies (Result =
              other)
```

feature -- Output

out: *STRING*
-- Printable representation of character
-- (From *GENERAL*.)

invariant

irreflexive_comparison: ***not*** (*Current* < *Current*)

end

5.7 Class *CHARACTER*

indexing

description: *"Characters, with comparison operations and an ASCII code"*

expanded class interface

CHARACTER

feature -- Access

code: *INTEGER*
-- Associated integer value

hash_code: *INTEGER*
-- Hash code value
-- (From *HASHABLE*.)

ensure

good_hash_value: *Result >= 0*

feature -- Comparison

infix *"<"* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is *other* greater than current character?
-- (From *COMPARABLE*.)

require

other_exists: *other /= Void*

ensure

asymmetric: *Result* ***implies not*** (*other < Current*)

infix *"<="* (*other*:***like*** *Current*): *BOOLEAN*
-- Is current character less than or equal to *other*?
-- (From *COMPARABLE*.)

require

other_exists: *other /= Void*

ensure

definition: *Result* = (*Current < other*) ***or*** *is_equal* (*other*);

infix *">="* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object greater than or equal to *other*?
-- (From *COMPARABLE*.)

require

other_exists: *other /= Void*

ensure

definition: *Result* = (*other <= Current*)

infix *">"* (*other*: ***like*** *Current*): *BOOLEAN*
-- Is current object greater than *other*?
-- (From *COMPARABLE*.)

require

other_exists: *other /= Void*

ensure

definition: *Result* = (*other < Current*)

max (*other*: ***like*** *Current*): ***like*** *Current*
-- The greater of current object and *other*
-- (From *COMPARABLE*.)

require

other_exists: *other /= Void*

ensure

current_if_not_smaller: (*Current >= other*) ***implies*** (*Result* = *Current*)

other_if_smaller: (*Current < other*) ***implies*** (*Result* = *other*)

min (*other*: ***like*** *Current*): ***like*** *Current*
-- The smaller of current object and *other*
-- (From *COMPARABLE*.)

require

other_exists: *other /= Void*

ensure

current_if_not_greater: (*Current <= other*) ***implies*** (*Result* = *Current*)

other_if_greater: (*Current > other*) ***implies*** (*Result* = *other*)

three_way_comparison (*other*: ***like*** *Current*): *INTEGER*
-- If current object equal to *other*, 0; if smaller,
-- –1; if greater, 1.
-- (From *COMPARABLE*.)

require

other_exists: *other /= Void*

ensure

equal_zero: (*Result = 0*) = *is_equal* (*other*);

smaller: (*Result = –1*) = *Current < other*;

greater_positive: (*Result = 1*) = *Current > other*

5.6 Class *BOOLEAN*

```
indexing
    description: "Truth values, with the boolean operations"
expanded class interface
    BOOLEAN
feature -- Access
    hash_code: INTEGER
            -- Hash code value
            -- (From HASHABLE.)
        ensure
            good_hash_value: Result >= 0
feature -- Basic operations
    infix "and" (other: BOOLEAN): BOOLEAN
            -- Boolean conjunction with other
        require
            other_exists: other /= Void
        ensure
            Result_exists: Result /= Void;
            de_morgan: Result = not (not Current or (not other));
            commutative: Result = (other and Current);
            consistent_with_semi_strict: Result implies (Current
              and then other)
    infix "and then" (other: BOOLEAN): BOOLEAN
            -- Boolean semi–strict conjunction with other
        require
            other_exists: other /= Void
        ensure
            Result_exists: Result /= Void;
            de_morgan: Result = not (not Current or else (not
              other));
    infix "implies" (other: BOOLEAN): BOOLEAN
            -- Boolean implication of other
            -- (semi–strict)
        require
            other_exists: other /= Void
        ensure
            definition: Result = (not Current or else other)
    prefix "not": BOOLEAN
            -- Negation.
    infix "or" (other: BOOLEAN): BOOLEAN
            -- Boolean disjunction with other
        require
            other_exists: other /= Void
        ensure
            Result_exists: Result /= Void;
            de_morgan: Result = not (not Current and (not other));
            commutative: Result = (other or Current);
            consistent_with_semi_strict: Result implies (Current or
              else other)
    infix "or else" (other: BOOLEAN): BOOLEAN
            -- Boolean semi–strict disjunction with other
        require
            other_exists: other /= Void
        ensure
            Result_exists: Result /= Void;
            de_morgan: Result = not (not Current and then (not
              other));
    infix "xor" (other: BOOLEAN): BOOLEAN
            -- Boolean exclusive or with other
        require
            other_exists: other /= Void
        ensure
            definition: Result = ((Current or other) and not (Current
              and other))
feature -- Output
    out: STRING
            -- Printable representation of boolean
invariant
    involutive_negation: is_equal (not (not Current));
    non_contradiction: not (Current and (not Current));
    completeness: Current or (not Current)
end
```

```
invariant
    neutral_addition: equal (Current + zero, Current);
    self_subtraction: equal (Current – Current, zero);
    neutral_multiplication: equal (Current * one, Current);
    self_division: divisible (Current) implies equal (Current /
        Current, one)
end
```

5.5 Class *NUMERIC*

```
indexing
    description: "Objects to which numerical operations are
        applicable";
    note: "The model is that of a commutative ring."
deferred class interface
    NUMERIC
feature -- Access
    one: like Current
            -- Neutral element for "*" and "/"
        deferred
        ensure
            Result_exists: Result /= Void
    zero: like Current
            -- Neutral element for "+" and "–"
        deferred
        ensure
            Result_exists: Result /= Void
feature -- Status report
    divisible (other: like Current): BOOLEAN
            -- May current object be divided by other?
        require
            other_exists: other /= Void
        deferred
    exponentiable (other: NUMERIC): BOOLEAN
            -- May current object be elevated to the power other?
        require
            other_exists: other /= Void
        deferred
feature -- Basic operations
    infix "+" (other: like Current): like Current
            -- Sum with other (commutative).
        require
            other_exists: other /= Void
        deferred
        ensure
            result_exists: Result /= Void;
            commutative: equal (Result, other + Current)
    infix "–" (other: like Current): like Current
            -- Result of subtracting other
        require
            other_exists: other /= Void
        deferred
        ensure
            result_exists: Result /= Void
    infix "*" (other: like Current): like Current
            -- Product by other
        require
            other_exists: other /= Void
        deferred
        ensure
            result_exists: Result /= Void
    infix "/" (other: like Current): like Current
            -- Division by other
        require
            other_exists: other /= Void;
            good_divisor: divisible (other)
        deferred
        ensure
            result_exists: Result /= Void
    infix "^" (other: NUMERIC): NUMERIC
            -- Current object to the power other
        require
            other_exists: other /= Void;
            good_exponent: exponentiable (other)
        deferred
        ensure
            result_exists: Result /= Void
    prefix "+": like Current
            -- Unary plus
        deferred
        ensure
            result_exists: Result /= Void
    prefix "–": like Current
            -- Unary minus
        deferred
        ensure
            result_exists: Result /= Void
```

5.4 Class *HASHABLE*

indexing

description: *"Values that may be hashed into an integer index, for use as keys in hash tables."*

note: *"In general, the default value of a type (e.g. 0 for integers, void for references) may not be hashed."*

deferred class interface

HASHABLE

feature -- Access

hash_code: *INTEGER*
-- Hash code value
deferred
ensure
good_hash_value: *Result >= 0*

end

5.3 Class *COMPARABLE*

```
indexing
	description: "Objects that may be compared according to a
		total order relation";
	note: "The basic operation is `<' (less than); others are defined
		in terms of this operation and `is_equal'."
deferred class interface
	COMPARABLE
feature -- Comparison
	infix "<" (other: like Current): BOOLEAN
			-- Is current object less than other?
		require
			other_exists: other /= Void
		deferred
		ensure
			asymmetric: Result implies not (other < Current)
	infix "<=" (other: like Current): BOOLEAN
			-- Is current object less than or equal to other?
		require
			other_exists: other /= Void
		ensure
			definition: Result = (Current < other) or is_equal
				(other);
	infix ">=" (other: like Current): BOOLEAN
			-- Is current object greater than or equal to other?
		require
			other_exists: other /= Void
		ensure
			definition: Result = (other <= Current)
	infix ">" (other: like Current): BOOLEAN
			-- Is current object greater than other?
		require
			other_exists: other /= Void
		ensure
			definition: Result = (other < Current)
	is_equal (other: like Current): BOOLEAN
			-- Is other attached to an object considered equal
			-- to current object?
			-- (Redefined from GENERAL.)
		require
			other_not_void: other /= Void
		ensure
			symmetric: Result implies other.is_equal (Current);
			consistent: standard_is_equal (other) implies Result;
			trichotomy: Result = (not (Current < other) and not
				(other < Current))
	max (other: like Current): like Current
			-- The greater of current object and other
		require
			other_exists: other /= Void
		ensure
			current_if_not_smaller: (Current >= other) implies
				(Result = Current)
			other_if_smaller: (Current < other) implies (Result =
				other)
	min (other: like Current): like Current
			-- The smaller of current object and other
		require
			other_exists: other /= Void
		ensure
			current_if_not_greater: (Current <= other) implies
				(Result = Current)
			other_if_greater: (Current > other) implies (Result =
				other)
	three_way_comparison (other: like Current): INTEGER)
			-- If current object equal to other, 0; if smaller,
			-- –1; if greater, 1.
		require
			other_exists: other /= Void
		ensure
			equal_zero: (Result = 0) = is_equal (other);
			smaller_negative: (Result = –1) = (Current < other);
			greater_positive: (Result = 1) = (Current > other)
invariant
	irreflexive_comparison: not (Current < Current)
end
```

5.2 Class *ANY*

indexing

description: *"Project–wide universal properties. This class is an ancestor to all developer-written classes. ANY inherits from GENERAL and may be customized for individual projects or teams."*

class interface

ANY

end

```
frozen standard_is_equal (other: like Current): BOOLEAN
        -- Is other attached to an object of the same type as
        -- current object, and field–by-field identical to it?
    require
        other_not_void: other /= Void
    ensure
        same_type: Result implies same_type (other);
        symmetric: Result implies other.standard_is_equal
          (Current)
feature -- Duplication
    frozen clone (other: GENERAL): like other
        -- Void if other is void; otherwise new object
        -- equal to other.
    ensure
        equal: equal (Result, other)
    copy (other: like Current)
        -- Update current object using fields of object attached
        -- to other, so as to yield equal objects.
    require
        other_not_void: other /= Void;
        type_identity: same_type (other)
    ensure
        is_equal: is_equal (other)
    frozen deep_clone (other: GENERAL): like other
        -- Void if other is void: otherwise, new object structure
        -- recursively duplicated from the one attached to other
    ensure
        deep_equal: deep_equal (other, Result)
    frozen standard_clone (other: GENERAL): like other
        -- Void if other is void; otherwise new object
        -- field–by-field identical to other.
        -- Always uses the default copying semantics.
    ensure
        equal: standard_equal (Result, other)
    frozen standard_copy (other: like Current)
        -- Copy every field of other onto corresponding field
        -- of current object.
    require
        other_not_void: other /= Void;
        type_identity: same_type (other)
    ensure
        is_standard_equal: standard_is_equal (other)
feature -- Basic operations
    frozen default: like Current
        -- Default value of current type
    frozen default_pointer: POINTER
        -- Default value of type POINTER
        -- (Avoid the need to write p.default for some p
        -- of type POINTER.)
    ensure
        Result = Result.default
    default_rescue
        -- Handle exception if no Rescue clause.
        -- (Default: do nothing.)
    frozen do_nothing
        -- Execute a null action.
    frozen Void: NONE
        -- Void reference
feature -- Output
    io: STD_FILES
        -- Handle to standard file setup
    out: STRING
        -- New string containing terse printable representation
        -- of current object
    print (some: GENERAL)
        -- Write terse external representation of some on
        -- standard output.
    frozen tagged_out: STRING
        -- New string containing printable representation of
        -- current object, each field preceded by its attribute
        -- name, a colon and a space.
invariant
    reflexive_equality: standard_is_equal (Current);
    reflexive_conformance: conforms_to (Current);
    involutive_object_id: id_object (object_id) = Current
end
```

5 SHORT FORMS OF REQUIRED CLASSES

5.1 Class *GENERAL*

indexing

description: *"Platform–independent universal properties. This class is an ancestor to all developer-written classes."*

class interface

GENERAL

feature -- Access

generating_type: *STRING*
-- Name of current object's generating type
-- (type of which it is a direct instance)

generator: *STRING*
-- Name of current object's generating class
-- (base class of the type of which it is a direct instance)

id_object (*id*: *INTEGER*): *ANY*
-- Object for which *object_id* has returned *id*;
-- void if none.

object_id: *INTEGER*
-- Value identifying current object uniquely;
-- meaningful only for reference types.

stripped (*other*: *GENERAL*): ***like*** *other*
-- New object with fields copied from current object,
-- but limited to attributes of type of *other*.

require

conformance: *conforms_to* (*other*)

ensure

stripped_to_other: *Result*.*same_type* (*other*)

feature -- Status report

frozen *conforms_to* (*other*: *GENERAL*): *BOOLEAN*
-- Does type of current object conform to type
-- of *other* (as per *Eiffel*: *The Language*, chapter13)?

require

other_not_void: *other* /= *Void*

frozen *same_type* (*other*: *GENERAL*): *BOOLEAN*
-- Is type of current object identical to type of *other*?

require

other_not_void: *other* /= *Void*

ensure

definition: *Result* = (*conforms_to* (*other*) ***and*** *other*.*conforms_to* (*Current*))

feature -- Comparison

frozen *deep_equal* (*some*: *GENERAL*; *other*: ***like*** *some*): *BOOLEAN*

-- Are *some* and *other* either both void

-- or attached to isomorphic object structures?

ensure

shallow_implies_deep: *standard_equal* (*some*, *other*) ***implies*** *Result;*

same_type: *Result* ***implies*** *some*.*same_type* (*other*);

symmetric: *Result* ***implies*** *deep_equal* (*other, some*)

frozen *equal* (*some*: *GENERAL*; *other*: ***like*** *some*): *BOOLEAN*

-- Are *some* and *other* either both void or attached

-- to objects considered equal?

ensure

definition: *Result* = (*some* = *Void* ***and*** *other* = *Void*) ***or else*** ((*some* /= *Void* ***and*** *other* /= *Void*) ***and then*** *some*.*is_equal* (*other*));

is_equal (*other*: ***like*** *Current*): *BOOLEAN*

-- Is *other* attached to an object considered equal

-- to current object?

require

other_not_void: *other* /= *Void*

ensure

consistent: *standard_is_equal* (*other*) ***implies*** *Result;*

same_type: Result implies same_type (other);

symmetric: *Result* ***implies*** *other*.*is_equal* (*Current*)

frozen *standard_equal* (*some*: *GENERAL; other*: ***like*** *some*): *BOOLEAN*

-- Are *some* and *other* either both void or attached to

-- field–by-field identical objects of the same type?

-- Always uses the default object comparison criterion.

ensure

definition: *Result* = (*some* = *Void* ***and*** *other* = *Void*) ***or else*** ((*some* /= *Void* ***and*** *other* /= *Void*) ***and then*** *some*.*standard_is_equal* (*other*))

4.16 • *INTEGER_REF* is a proper descendant of *NUMERIC*.

4.17 • *POINTER* is a proper descendant of *POINTER_REF*.

4.18 • *POINTER_REF* is a proper descendant of *HASHABLE*.

4.19 • *REAL* is a proper descendant of *REAL_REF*.

4.20 • *REAL_REF* is a proper descendant of *COMPARABLE*.

4.21 • *REAL_REF* is a proper descendant of *HASHABLE*.

4.22 • *STRING* is a proper descendant of *COMPARABLE*.

4.23 • *STRING* is a proper descendant of *HASHABLE*.

4.24 • *STRING* is a proper descendant of *HASHABLE*.

[4.1 follows from *Eiffel: The Language*; the language description is considered to be amended in such a way that *PLATFORM* is a class without privileges, to be inherited explicitly by classes which need access to its features.]

3.17 • *MEMORY* [flatshort form in section 5.17].

3.18 • *EXCEPTIONS* [flatshort form in section 5.18].

3.19 • *ARGUMENTS* [flatshort form in section 5.19].

3.20 • *PLATFORM* [flatshort form in section 5.20].

3.21 • *BOOLEAN_REF* [flatshort form in section 5.21].

3.22 • *CHARACTER_REF* [flatshort form in section 5.22].

3.23 • *DOUBLE_REF* [flatshort form in section 5.23].

3.24 • *INTEGER_REF* [flatshort form in section 5.24].

3.25 • *POINTER_REF* [flatshort form in section 5.25].

3.26 • *REAL_REF* [flatshort form in section 5.26].

[The classes appear in this section and section and section 5 in the following order: universal classes; deferred classes for basic classes; basic classes; arrays and strings; operating system interface; auxiliary reference classes for the definition of basic classes.]

4 REQUIRED ANCESTRY LINKS

The following constitute the required ancestry links [ordered alphabetically, after the first rule, by the name of the applicable descendant class]:

4.1 • Every Required Class except *GENERAL* is a descendant of *ANY*

4.2 • *ANY* is an heir of *GENERAL*.

4.3 • *BOOLEAN* is a proper descendant of *BOOLEAN_REF*.

4.4 • *BOOLEAN_REF* is a proper descendant of *HASHABLE*.

4.5 • *CHARACTER* is a proper descendant of *CHARACTER_REF*.

4.6 • *CHARACTER_REF* is a proper descendant of *COMPARABLE*.

4.7 • *CHARACTER_REF* is a proper descendant of *HASHABLE*.

4.8 • *DOUBLE* is a proper descendant of *DOUBLE_REF*.

4.9 • *DOUBLE_REF* is a proper descendant of *COMPARABLE*.

4.10 • *DOUBLE_REF* is a proper descendant of *HASHABLE*.

4.11 • *DOUBLE_REF* is a proper descendant of *NUMERIC*.

4.12 • *FILE* is a proper descendant of *MEMORY*.

4.13 • *INTEGER* is a proper descendant of *INTEGER_REF*.

4.14 • *INTEGER_REF* is a proper descendant of *COMPARABLE*.

4.15 • *INTEGER_REF* is a proper descendant of *HASHABLE*.

2.4.1.8 • For a feature that appears in both *ic* and *sc*, an argument type in *ic* that is different from the corresponding type in *sc* but conforms to it.

2.4.1.9 • For a feature that appears in both *ic* and *sc*, a line that appears in the Header_comment of *ic* but not in that of *sc*.

2.4.1.10 • An Index_clause that appears in *ic* but not in *sc*.

2.4.1.11 • A difference regarding the order in which a feature appears in *ic* and *sc*, the Feature_clause to which it belongs, the Header_comment of such a Feature_clause, or the presence in *ic* of a Feature_clause that has no counterpart in *sc*.

[As a consequence of section 2.4.1.11, the division of classes into one Feature_clause or more, and the labels of these clauses, appear in this document for the sole purpose of readability and ease of opdreference, but are not part of this Standard.]

[The goal pursued by the preceding definition is to make sure that an Eiffel system that follows this Standard will be correctly processed by any compatible implementation, without limiting the implementors' freedom to provide more ambitious facilities.]

3 REQUIRED CLASSES

The Required Classes are the following twenty classes [ordered from the general to the specific, as in section 5]:

3.1 • *GENERAL* [flatshort form in section 5.1].

3.2 • *ANY* [flatshort form in section 5.2].

3.3 • *COMPARABLE* [flatshort form in section 5.3].

3.4 • *HASHABLE* [flatshort form in section 5.4].

3.5 • *NUMERIC* [flatshort form in section 5.5].

3.6 • *BOOLEAN* [flatshort form in section 5.6].

3.7 • *CHARACTER* [flatshort form in section 5.7].

3.8 • *INTEGER* [flatshort form in section 5.8].

3.9 • *REAL* [flatshort form in section 5.9].

3.10 • *DOUBLE* [flatshort form in section 5.10].

3.11 • *POINTER* [flatshort form in section 5.10].

3.12 • *ARRAY* [flatshort form in section 5.12].

3.13 • *STRING* [flatshort form in section5.13).

3.14 • *STD_FILES* [flatshort form in section 5.14].

3.15 • *FILE* [flatshort form in section 5.15].

3.16 • *STORABLE* [flatshort form in section 5.16].

2.3.1.5 • The header comment of any inherited feature coming from a Required Class *A* and having the same name in *C* as in *A* shall end with a line of the form:

-- (From *A*.)

2.3.1.6 • The header comment of any inherited feature coming from a Required Class *A* and having a name in *C* different from its name *x* in *A* shall end with a line of the form:

-- (From *x* in *A*.)

[The comments defined in the last two rules are applicable regardless of whether *C* redeclares the feature.]

2.3.1.7 • If deferred, *C* shall appear as ***deferred class***.

2.3.1.8 • Any deferred feature of *C* shall be marked as ***deferred***.

2.3.1.9 • In case of precondition redeclaration, the successive preconditions shall appear as a single Precondition clause, separated by semicolons.

2.3.1.10 • In case of postcondition redeclaration, the successive preconditions shall appear as a single Postcondition clause, separated by ***and then***.

2.4 Flatshort Compatibility

2.4.1 Definition

A class appearing in an Eiffel implementation is said to be Flatshort-Compatible with a class of the same name listed in this Standard if and only if any difference that may exist between its flatshort form *ic* and the flatshort form *sc* of the corresponding class as it appears in section 5, where both flatshort forms follow the conventions of section 2.3, belongs to one of the following eleven categories:

2.4.1.1 • A feature that appears in *ic* but not in *sc*, whose Header_comment includes, as its last line, the mention:

-- (Feature not in Kernel Library Standard.)

2.4.1.2 • An invariant clause that appears in *ic* but not in *sc*.

2.4.1.3 • For a feature that appears in both *ic* and *sc*, a postcondition clause that appears in *ic* but not in *sc*.

2.4.1.4 • For a feature that appears in both *ic* and *sc*, a precondition in *sc* that implies the precondition in *ic*, where the implication is readily provable using rules of mathematical logic.

2.4.1.5 • For a feature that appears in both *ic* and *sc*, a postcondition or invariant clause in *ic* that implies the corresponding clause in *sc*, where the implication is readily provable using rules of mathematical logic.

2.4.1.6 • A difference between the Tag_mark of an Assertion_clause in *ic* and its counterpart in *sc*.

2.4.1.7 • For a feature that appears in both *ic* and *sc*, an argument type in *sc* that is different from the corresponding type in *ic* but conforms to it.

2.2 Kernel compatibility

2.2.1 Definition

An Eiffel implementation will be said to be kernel-compatible if and only if it includes a set of classes satisfying the following five conditions:

2.2.1.1 • For each of the Required Classes, the implementation includes a class with the same name.

2.2.1.2 • All the Required Ancestry Links are present between these classes.

2.2.1.3 • The flatshort form of each one of these classes is Flatshort-Compatible with the corresponding Required Flatshort Form.

2.2.1.4 • All the dependents of the Required Classes in the implementation are also included in the implementation.

2.2.1.5 • None of the features appearing in the Required Flatshort Forms appears in a Rename clause of any of the implementation's Required Classes.

[These conditions allow a kernel-compatible implementation to include inheritance links other than the ones described in this Standard; condition 2.2.1.4 indicates that for any such link the additional proper ancestors must also be provided by the implementors, since the dependents of a class include its parents.]

[Condition 2.2.1.4 guarantees that if a feature name appears in this Standard both in the Flatshort form of a Required Class and in the flatshort form of one of its proper ancestors, it corresponds to the same feature or to a redefinition of it.]

2.3 Flatshort Conventions

2.3.1 Definition

In the process of assessing for Flatshort Compatibility a class *C* from a candidate implementation, the following ten conventions, which have been applied to the Required Flatshort Forms as they appear in this Standard, shall be applied:

2.3.1.1 • No feature shall be included unless it is generally available (as defined in *Eiffel: The Language*, page 100) or is a general creation procedure (as defined in *Eiffel: The Language*, page 285).

2.3.1.2 • The Creation clause of the flatshort specification shall include the full specification of all general creation procedures of *C*.

2.3.1.3 • Any feature of *C* not inherited from *GENERAL* shall be included in one of the Feature clauses.

[As a consequence of the last two rules the specification of a creation procedure that is also generally exported will appear twice: in the Creation clause and in a Feature clause. Also note that the "features of a class" include inherited as well as immediate features, so that all features inherited from an ancestor other than *GENERAL* must appear in the flatshort form.]

2.3.1.4 • A feature *f* from *GENERAL* shall be included if and only if *C* redeclares *f*.

1.2 Scope of this Standard

This Standard defines a number of library-related conditions that an Eiffel implementation must satisfy. These conditions affect a set of classes known as the kernel library. An implementation that satisfies the conditions described in this Standard will be said to be **kernel-compatible**, a phrase that is abbreviated in this Standard as just “compatible”.

[In other contexts it may be preferable to use the full phrase, since the compatibility of an Eiffel implementation also involves other aspects, such as language compatibility.]

[The terms “compatibility” and “compatible” may be felt to be less clear than “conformance” and “conformant”. The former are used here, however, since talking about conformance might cause confusions with the Eiffel notion of a type conforming to another.]

1.3 Other documents

The phrase *Eiffel: The Language* as used in this Standard refers to the second printing of the book *Eiffel: The Language*, Prentice Hall, 1992, ISBN 0-13-245-925-7.

For the purposes of this Standard, the definition of the Eiffel language is the definition given by *Eiffel: The Language*.

In case of contradictions between the library specifications given by *Eiffel: The Language* and those given in this Standard, the latter shall take precedence.

2 COMPATIBILITY CONDITIONS

2.1 Definitions

2.1.1 Required Classes

In this Standard, the phrase “Required Classes” denotes a set of classes whose names are those listed in section 3.

2.1.2 Required Flatshort Form

In this Standard, the phrase “Required Flatshort Forms” denotes the flatshort forms given for the Required Classes in section 3.

2.1.3 Flatshort Compatibility

In this Standard, a class is said to be Flatshort-Compatible with one of the short forms given in this Standard if it satisfies the conditions given in section 2 of this Standard.

2.1.4 Required Ancestry Links

In this Standard, the expression “Required Ancestry Links” denotes the inheritance links specified in section 4 of this Standard.

[The term “Ancestry” is used rather than “Inheritance” because the required links may be implemented by indirect rather than direct inheritance, except for *ANY* which must be a direct heir of *GENERAL* as per rule 4.2, given on page 10.]

0 INTRODUCTION

[This introduction is not part of the Standard.]

0.1

To favor the interoperability between implementations of Eiffel, it is necessary, along with a precise definition of the language, to have a well-defined set of libraries covering needs that are likely to arise in most applications. This library is known in Eiffel as the Kernel Library.

0.2

The present document defines a standard for the Kernel Library. If an Eiffel implementation satisfies this Standard — under the precise definition of *Kernel Compatibility* given in section 2.2 — it will be able to handle properly any Eiffel system whose use of the Kernel Library only assumes the library properties defined in this Standard.

0.3

The Eiffel Library standardization process, as described in Appendix A of the present document, is based on a dynamic view which, in the spirit of Eiffel's own "feature obsolescence" mechanism, recognizes the need to support evolution while preserving the technology investment of Eiffel users. One of the consequences of this dynamic view is to define *vintages* corresponding to successive improvements of the Standard. The present document describes **Vintage 95**, valid for the calendar year 1995.

1 CONTENTS OF THIS STANDARD

1.1 Definition: this Standard

The Eiffel Kernel Library Standard, denoted in the present document by the phrase "this Standard", is made up of the contents of sections 1 to 5 of the present document, with the exception of elements appearing between square brackets [...] which are comments.

[As a result of the preceding definition the following elements are not part of this Standard: section 0, the table of contents, Appendix A in section 6 (the Kernel Library Standardization process), Appendix B in section 7 (list of differences), the Index in section 8, and elements playing a pure typesetting role such as page headers.]

CONTENTS

Report identification

TR-EI-48/KL: *The Eiffel Library Standard.* (Earlier title: *The Proposed Eiffel Library Kernel Standard*). Prepared for the Nonprofit International Consortium for Eiffel by:

Interactive Software Engineering Inc.
270 Storke Road, Suite 7
Goleta, CA 93117 USA
Phone 805-685-1006, Fax 805-685-6869
Electronic Mail <info@eiffel.com>

Publication history

First published: 14 July 1994.

Version 5: 25 September 1994.

Version 6: 10 October 1994.

Version 7: 12 January 1995, integrating the result of the votes of the NICE library committee (December 1994) on classes *ANY*, *PLATFORM and GENERAL*..

The present document is version 8, 4 June 1995, designed as vintage 95 library standard.

Authors

The document was written by Bertrand Meyer and for versions up to 7 benefited from numerous discussions with Michael Schweitzer.

The substance (the kernel specification) is the result of a collective effort with contributions from:

Eric Bezault (ISE, Santa Barbara)
Roger Browne (Everything Eiffel, Great Britain)*
Didier Dupont (SOL, Paris)
Fred Hart (Tower Technology, Atlanta)*
Aaron Hillegas (Tower Technology, Atlanta)
Rock Howard (Tower Technology, Austin)*
Paul Johnson (GEC-Marconi, Great Britain)**
Xavier Le Vourch (ISE, Santa Barbara)
David Morgan (ISE, Santa Barbara)
Bertrand Meyer (ISE, Santa Barbara)
Christine Mingins (Monash University, Melbourne), chairman of the NICE library committee
Michael Schweitzer (Swissoft, Göttingen)*
Philippe Stephan (SCENEC, Paris)
Robert Switzer (Swissoft, Göttingen, and University of Göttingen)*
Steve Tynor (Tower Technology, Atlanta)*
Dino Valente (ISE, Santa Barbara)

The influence of the Gustave kernel proposal (authors marked * above) and the GEC-Marconi kernel proposal (author marked ** above) is gratefully acknowledged.

THE EIFFEL LIBRARY STANDARD

VINTAGE 95