NICE ELKS 2000 proposal

Dominique Colnet, Emmanuel Stapf, Olivier Zendra

NICE ELKS 2000

Welcome! This official NICE page hosts the ELKS 2000 initial and current proposals.

In 1995, NICE (The Non-Profit International Consortium for Eiffel) adopted the ELKS'95 standard for Eiffel kernel libraries. Since then, and despite the fact it was supposed to be updated each year and although it was considered by many as an imperfect standard, ELKS had not evolved beyond ELKS'95.

In July 1999, Emmanuel Stapf (from ISE) and Dominique Colnet and Olivier Zendra (both from the SmallEiffel team) met in Nancy, France, and worked on a much-needed proposal (limited to ARRAY and STRING) to try and get ELKS evolving a bit. This initial ELKS 2000 draft proposal was then submitted for review to other Eiffel experts, then to a meeting in TOOLS USA.

It is now discussed in the NICE mailing list dedicated to library issues, http://www.egroups.com/group/eiffel-nice-library.

This site gives acces to the following ressources:

current_array.html
The current state of the ELKS 2000 proposal for ARRAY.

current_string.html
The current state of the ELKS 2000 proposal for STRING.

elks2000init.html
The initial ELKS 2000 proposal.

inital_array.html
The initial ELKS 2000 proposal for ARRAY.

inital_string.html
The initial ELKS 2000 proposal for STRING.

changelog.html
The list of changes between the initial and the current ELKS 2000 proposals.

Back to the top

Maintainers: Dominique Colnet and Olivier Zendra.

Last Update: 20 August 1999.

5.12 Class ARRAY (current ELKS2000 status)

[Please don't report broken URLs because this page is still a draft.]

```
indexing

    description: "Resizable 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.)
        require
            valid_indices: (minindex <= maxindex) or (maxindex = minindex - 1)
        ensure
            lower_set: lower = minindex
            upper_set: upper = maxindex
            -- array_set: for i in minindex..maxindex, item(i) = default item

    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.)
       require
            valid_array: a /= Void

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.)
        require
            valid_indices: (minindex <= maxindex) or (maxindex = minindex - 1)
        ensure
            lower_set: lower = minindex
            upper_set: upper = maxindex
            -- array_set: for i in minindex..maxindex, item(i) = default item

    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.)
       require
            valid_array: a /= Void

feature -- Access

    first: like item
            -- First element
        require
            not_empty: not is_empty
        ensure
            definition: Result = item (lower)

    has (v: like item): BOOLEAN
            -- Does Current contain v?
            -- Use equal as comparison operator.
        ensure
            definition: Result = valid_index(index_of(c));
```

```
    index_of (v: like item): INTEGER
            -- Index of first occurrence of v
            -- Use equal as comparison operator.
            -- Upper + 1 if not found
        ensure
            not_found: (Result = upper + 1) = not has(v);
            found: has(v) implies equal(v,item(Result));

    infix "@", item (i: INTEGER): G
            -- Entry at index i
        require
            good_key: valid_index (i)

    last: like item
            -- Last element
        require
            not_empty: not is_empty
        ensure
            definition: Result = item (upper)

    reverse_index_of (v: like item): INTEGER
            -- Index of last occurrence of v?
            -- Use equal as comparison operator.
            -- Lower - 1 if not found
        ensure
            not_found: (Result = upper + 1) = not has(v);
            found: has(v) implies equal(v,item(Result));

feature -- Measurement

    capacity: INTEGER
            -- Number of elements that Current can currently contain in the internal storag
            -- Automatically updated when resizing is needed. Useful for fine tuning.

    count: INTEGER
            -- Number of valid indices
        ensure
            definition: Result = upper - lower + 1;

    lower: INTEGER
            -- Minimum valid index

    occurrences (v: like item): INTEGER
            -- Number of times v appears in Current.
            -- Use equal as comparison operator.
        ensure
            non_negative_occurrences: Result >= 0

    upper: INTEGER
            -- Maximum valid index

feature -- Comparison

    is_equal (other: like Current): BOOLEAN
            -- Do both arrays have same lower, upper and
            -- items (compared with is_equal)?
            -- (Redefined from GENERAL.)

feature -- Status report

    is_empty: BOOLEAN
            -- Is array empty?
        ensure
            definition: Result = (count = 0)

    valid_index (i: INTEGER): BOOLEAN
            -- Is i within the bounds of the array?
        ensure
            definition: lower <= i and i <= upper

feature -- Element change

    add (v: like item; i: INTEGER)
        -- Insert v at index i, shifting elements between ranks i and upper rightwards.
```

```
            require
                valid_insertion_index: lower <= i and i <= upper + 1
            ensure
                inserted: item (i) = v;
                same_lower: lower = old lower;
                higher_upper: upper = old upper + 1

        add_first (v: like item)
                -- Insert v at index lower, shifting all elements rightwards.
            ensure
                inserted: first = v;
                same_lower: lower = old lower;
                higher_upper: upper = old upper + 1

        add_last (v: like item)
                -- Append v at end.
            ensure
                inserted: last = v;
                same_lower: lower = old lower;
                higher_upper: upper = old upper + 1

        force (v: like item; i: INTEGER)
                -- Assign v to i-th entry.
                -- Always applicable: resize the array if i falls out of
                -- currently defined bounds; preserve existing items.
                -- New entries if any are initialized to default value.
            ensure
                put: item (i) = v;
                higher_count: count >= old count

        put (v: like item; i: INTEGER)
                -- Assign v to i-th entry.
            require
                good_index: valid_index (i)
            ensure
                put: item (i) = v

    feature -- Removal

        remove (i: INTEGER)
                -- Remove element at index i and shift elements between ranks i+1 and upper lef
            require
                good_index: valid_index (i)
            ensure
                same_lower: lower = old lower;
                smaller_upper: upper = old upper - 1

        remove_first
                -- Remove first element and shift all elements leftwards.
            require
                not_is_empty: not is_empty
            ensure
                same_lower: lower = old lower;
                smaller_upper: upper = old upper - 1

        remove_last
                -- Remove last element.
            require
                not_is_empty: not is_empty
            ensure
                same_lower: lower = old lower;
                smaller_upper: upper = old upper - 1

        wipe_out
                -- Remove all elements.
            ensure
                is_empty: is_empty
                same_capacity: capacity = old capacity
                same_lower: lower = old lower

    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 in the
            -- intersection between [old lower .. old upper] and
            -- [minindex .. maxindex].
            -- New positions are initialized to their default value.
        require
            valid_indices: (minindex <= maxindex) or (maxindex = minindex - 1)
        ensure
            lower_set: lower = minindex
            upper_set: upper = maxindex

feature -- Conversion

    to_external: POINTER
            -- Address of the storage area containing the actual sequence of elements,
            -- to be passed to some external (non-Eiffel) routine.
            -- Keep in mind that this storage area is subject to garbage collection.

feature -- Duplication

    copy (other: like Current)
            -- Reinitialize by copying all the items of other.
            -- Shallow copy: if the elements are references, only the references
            -- are copied, not the object they point to.
            -- (Redefined from GENERAL.)
        ensure
            same_lower: lower = other.lower;
            same_upper: upper = other.upper;
            -- same_content: For every i in lower..upper,
            --     item (i) = other.item (i)

feature -- Indexing

    reindex (new_lower: INTEGER)
            -- Change bounds according to new_lower, keeping same size and contents.
        ensure
            reindexed_lower: lower = new_lower;
            same_contents: count = old count;

feature -- Optimization

    adjust_capacity (n: INTEGER)
            -- Rearrange array so that it can accommodate
            -- at least n elements.
            -- Do not loose any previously entered element.
        require
            no_item_lost: n >= count
        ensure
            enough_capacity: capacity >= n

invariant

    consistent_count: count = upper - lower + 1;
    non_negative_count: count >= 0;
    good_capacity: capacity >= count

end
```

Last Updated: 01 September 1999

5.13 Class STRING (current ELKS2000 status)

[Please don't report broken URLs because this page is still a draft.]

```
indexing

    description: "Resizable sequences of characters, accessible through %
        %integer indices in a contiguous range. If not empty, first element%
        %is at index 1."

class interface

    STRING

creation

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

    make_from_external (ptr: POINTER)
            -- Set Current with a copy of the content of ptr.
            -- Assume ptr is a null-terminated memory area containing
            -- only characters.
            -- The extra null character is not part of the Eiffel string
        require
            ptr_exists: ptr /= default_pointer

    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
        ensure
            count_set: count = s.count

feature -- Initialization

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

    make_from_external (ptr: POINTER)
            -- Set Current with a copy of the content of ptr.
            -- Assume ptr is a null-terminated memory area containing
            -- only characters.
            -- The extra null character is not part of the Eiffel string
        require
            ptr_exists: ptr /= default_pointer

    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
        ensure
            count_set: count = s.count

feature -- Access
```

```
first: CHARACTER
        -- First character of Current.
    require
        not_empty_string: not is_empty
    ensure
        good_result: Result = item (1)

has (c: CHARACTER): BOOLEAN
        -- Does Current contain c ?
    ensure
        definition: Result = (index_of(c) /= 0);

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

index_from (c: CHARACTER; start_index: INTEGER): INTEGER
        -- Index of first occurrence of c at or after start_index;
        -- 0 if none.
    require
        valid_start_index: valid_index (start_index)
    ensure
        non_negative_result: Result >= 0;
        valid_result: Result > 0 implies valid_index (Result)
        result_after_start_index: Result > 0 implies Result >= start_index;
        at_this_index: Result > 0 implies item (Result) = c;
        -- none_before: For every i in start_index..Result-1, item (i) /= c
        -- zero_iff_absent:
        --     (Result = 0) = For every i in start_index..count, item (i) /= c

index_of (c: CHARACTER): INTEGER
        -- Index of first occurrence of c;
        -- 0 if none.
    ensure
        empty_definition: empty implies Result = 0;
        non_empty_definition: not empty implies Result = index_from (c, 1);

infix "@", item (i: INTEGER): CHARACTER
        -- Character at index i
    require
        good_key: valid_index (i)

last: CHARACTER
        -- Last character of Current.
    require
        not_empty_string: not is_empty
    ensure
        good_result: Result = item (count)

reverse_index_from (c: CHARACTER; last_index: INTEGER): INTEGER
        -- Index of first occurrence of c at or before last_index;
        -- 0 if none.
    require
        valid_last_index: valid_index (last_index)
    ensure
        non_negative_result: Result >= 0;
        valid_result: Result > 0 implies valid_index(Result)
        result_before_last_index: Result <= last_index;
        at_this_index: Result > 0 implies item (Result) = c;
        -- none_before: Result > 0 implies For every i in Result+1..last_index, item (i
        -- zero_iff_absent:
        --     (Result = 0) = For every i in 1..last_index, item (i) /= c

reverse_index_of (c: CHARACTER): INTEGER
        -- Index of last occurrence of c;
        -- 0 if none.
    ensure
        empty_definition: empty implies Result = 0;
        non_empty_definition: empty implies Result = reverse_index_from (c, count);

reverse_string_index_from (other: STRING; last_index: INTEGER): INTEGER
```

```
            -- Index of first occurrence of other at or before last_index;
            -- 0 if none.
        require
            other_not_empty: not other.empty;
            valid_last_index: valid_index (last_index)
        ensure
            non_negative_result: Result >= 0;
            valid_result: Result > 0 implies valid_index (Result)
            result_before_last_index: Result > 0 implies Result <= last_index;
            at_this_index: Result > 0 implies other.is_equal (substring (Result, Result - 1
            -- none_before: Result > 0 implies
            --     For every i in Result+1..last_index, not other.is_equal (substring (i, i
            -- zero_iff_absent:
            --     (Result = 0) = For every i in 1..last_index, not other.is_equal (substri

    reverse_string_index_of (other: STRING): INTEGER
            -- Index of last occurrence of other;
            -- 0 if none.
        require
            other_not_empty: not other.empty
        ensure
            empty_definition: empty implies Result = 0;
            non_empty_definition: not empty implies Result = reverse_string_index_from (oth

    string_index_from (other: STRING; start_index: INTEGER): INTEGER
            -- Index of first occurrence of other at or after start;
            -- 0 if none.
        require
            other_not_empty: not other.empty;
            valid_start_index: valid_index (start_index)
        ensure
            non_negative_result: Result >= 0;
            valid_result: Result > 0 implies valid_index (Result)
            result_after_start_index: Result > 0 implies Result >= start_index;
            at_this_index: Result > 0 implies other.is_equal (substring (Result, Result - 1
            -- none_before:
            --     For every i in start_index..Result-1, not other.is_equal (substring (i, 
            -- zero_iff_absent:
            --     (Result = 0) = For every i in start_index..count, not other.is_equal (su

    string_index_of (other: STRING): INTEGER
            -- Index of first occurrence of other;
            -- 0 if none.
        require
            other_not_empty: not other.empty
        ensure
            empty_definition: empty implies Result = 0;
            non_empty_definition: not empty implies Result = reverse_string_index_from (oth

feature -- Measurement

    count: INTEGER
            -- Current number of characters making up the string

    capacity: INTEGER
            -- Number of characters that Current can currently contain in the internal stora
            -- Automatically updated when resizing is needed. Useful for fine tuning.

    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

    infix "<" (other: like Current): 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)

    same_as (other: STRING): BOOLEAN
            -- Is string made of same character sequence as other?
            -- Case insensitive comparison.
        require
            other_exists: other /= Void

    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

    is_empty: BOOLEAN
            -- Is string empty?
        ensure
            definition: Result = (count = 0)
```

```
    valid_index (index: INTEGER): BOOLEAN
            -- Is i within the bounds of the string?
        ensure
            definition: Result = (1 <= index and index <= count)

    is_boolean: BOOLEAN
            -- Is Current representing a boolean ("true" or "false" with no
            -- case consideration)?

    is_double: BOOLEAN
            -- Can Current be read as a DOUBLE?

    is_integer: BOOLEAN
            -- Can Current be read as an INTEGER?

    is_real: BOOLEAN
            -- Can Current be read as a REAL?

feature -- Element change

    add, insert (c: CHARACTER; i: INTEGER)
            -- Insert c at index i, shifting characters between ranks i and count rightwards
        require
            valid_insertion_index: 1 <= i and i <= count + 1
        ensure
             inserted: item (i) = c
             one_more_character: count = old count + 1

    add_first, precede (c: CHARACTER)
            -- Prepend c at beginning.
        ensure
            inserted: first = c
            one_more_character: count = old count + 1

    add_last, append_character, extend (c: CHARACTER)
            -- Append c at end.
        ensure
            inserted: last = c
            one_more_character: count = old count + 1

    append, append_string (s: STRING)
            -- Append a copy of s at end.
        require
            other_exists: s /= Void
        ensure
            new_count: count = old count + old s.count
            -- appended: For every i in 1.. old s.count,
            --     item (old count + i) = s.item (i)

    append_boolean (b: BOOLEAN)
            -- Append the string representation of b at end.
        ensure
            boolean_inserted: reverse_index_of (b.out) = count - b.out.count
            new_count: count = old count + b.out.count + 1

    append_double (d: DOUBLE)
            -- Append the string representation of d at end.
        ensure
            double_inserted: reverse_index_of (d.out) = count - d.out.count
            new_count: count = old count + d.out.count + 1

    append_integer (i: INTEGER)
            -- Append the string representation of i at end.
        ensure
            integer_inserted: reverse_index_of (i.out) = count - i.out.count
            new_count: count = old count + i.out.count + 1

    append_real (r: REAL)
            -- Append the string representation of r at end.
        ensure
            real_inserted: reverse_index_of (r.out) = count - r.out.count
            new_count: count = old count + r.out.count + 1

    fill (c: CHARACTER)
```

```
            -- Replace every character with c.
        ensure
            same_count: old count = count
            filled: occurrences(c) = count

    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_string (s: like Current; i: INTEGER)
            -- Insert s at index i, shifting characters between ranks i and count rightwards
        require
            string_exists: s /= Void;
            valid_insertion_index: 1 <= i and i <= count + 1
        ensure
            new_count: count = old count + old s.count
            inserted: s.is_equal (substring (i, i - 1 + old s.count))

    left_adjust
            -- Remove leading white space.
        ensure
            shorter: count <= old count
            first_not_white_space: not is_empty implies ("%T%R%N ").index_of (first) = 0

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

    right_adjust
            -- Remove trailing white space.
        ensure
            shorter: count <= old count
            last_not_white_space: not is_empty implies ("%T%R%N ").index_of (last) = 0

    subcopy (other: like Current; start_pos, end_pos, from_index: INTEGER) is
            -- Copy characters of other within bounds start_pos and
            -- end_pos to current string starting at index from_index.
        require
            other_not_void: other /= Void;
            valid_start_pos: other.valid_index (start_pos)
            valid_end_pos: other.valid_index (end_pos)
            valid_bounds: start_pos <= end_pos
            valid_from_index: valid_index (from_index)
            enough_space: valid_index (from_index + end_pos - start_pos)
        ensure
            same_count: old count = count
            copied: substring (from_index, from_index + end_pos - start_pos).is_equal
                    (other.substring (start_pos, end_pos))

    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, shifting characters between ranks i + 1 and count le
        require
            valid_removal_index: valid_index (i)
        ensure
            new_count: count = old count - 1
```

```
    wipe_out
            -- Remove all characters.
        ensure
            empty_string: is_empty
            same_capacity: capacity = old capacity
            same_lower: lower = old lower

feature -- Conversion

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

    to_double: DOUBLE
            -- Double value; for example, when applied to "123.0",
            -- will yield 123.0 (double)
        require
            is_double: is_integer or is_real or is_double

    to_external: POINTER
            -- Address of the storage area containing the actual sequence of characters,
            -- to be passed to some external (non-Eiffel) routine.
            -- Keep in mind that this storage area is subject to garbage collection.

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

    to_lower
            -- Convert to lower case.

    to_real: REAL
            -- Real value;
            -- for example, when applied to "123.0", will yield 123.0
        require
            is_real: is_integer or is_real

    to_upper
            -- Convert to upper case.

feature -- Duplication

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

    infix "+" (other: STRING): like Current
            -- Create a new object which is the concatenation of Current and other.
        require
            other_exists: other /= Void
        ensure
            result_count: Result.count = count + other.count

    substring (start_index, end_index: INTEGER): like Current
            -- Create a substring containing all characters from start_index
            -- to end_pos included.
        require
            valid_start_index: valid_index (start_index)
            valid_end_index: valid_index (end_index)
            meaningful_interval: start_index <= end_index
        ensure
            new_result_count: Result.count = end_index - start_index + 1
            -- original_characters: For every i in 1..end_index - start_index + 1,
            --     Result.item (i) = item (start_index + i -1)
```

```
feature -- Output

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

feature -- Optimization

    adjust_capacity (n: INTEGER)
            -- Rearrange string so that it can accommodate
            -- at least n characters.
            -- Do not lose any previously entered character.
        require
            no_item_lost: n >= count
        ensure
            enough_capacity: capacity >= n

invariant

    non_negative_count: count >= 0
    capacity_big_enough: capacity >= count

end
```

Last Updated: 15 September 1999

ELKS 2000 initial proposal

Content

Preamble

This work is a **proposal** for the new standard ELKS for two classes: ARRAY and STRING.

It is not intended to be the whole new standard, but only **part of it**. Indeed, because ELKS hasn't evolved at all since 1995, we decided to invest a reasonnable but limited amount of time and work to see whether it was possible to make things go ahead a bit.

If, **after discussions and possibly modifications**, this proposal for ARRAY and STRING is **officially accepted in a reasonnable amount of time and published** by NICE as part of the new standard (which would be the first evolution since 1995...), we will continue working on other classes and topics for the new standard. Otherwise, we will, as most people so far who have been trying to improve ELKS, consider we have wasted our time and efforts and give up.

This work is mostly considered as a (major) **cleanup of ELKS'95 for ARRAY and STRING**, not as an extension to it. Once good basis are agreed on, extensions may be discussed more easily.

(This preamble was written by Dominique Colnet and Olivier Zendra, from the SmallEiffel team. As far as we know, these views are shared by Emmanuel Stapf, from ISE. However, since he could not take part in writing this, neither him or ISE are committed by this.)

In Progress for the new standard

STRING (Note: this list may be incomplete. initial_string.html is the actual reference.)

1. Introduced *capacity*.
2. *make* is not frozen anymore.
3. Removed *remake* and *resize*.
4. Renamed *empty* into *is_empty*.
5. Reviewed most of the assertions to use *valid_index* instead of the explicite use of the comparison of index to 1 and count.
6. Renamed *from_c* into *make_from_external*
7. Added a new creation procedure *make_from_external* and *to_external*
8. Introduced a new set of routines to do some string search based on character/string search from with/without a start/last index which can do the search from the beginning or from the end:
    - *index_from*
    - *index_of*
    - *reverse_index_from*
    - *reverse_string_index_of*
    - *reverse_index_of*
    - *reverse_string_index_from*
    - *string_index_from*
    - *string_index_of*
9. Changed the post-conditions of *left_adjust* and *right_adjust* which were completely wrong.
10. Removed *put_substring* and replace it by *subcopy* a multipurpose string routine manipulation with a clear comment.
11. Changed *insert* and *insert_character* into *insert_string* and *insert*. Clarified the purpose of those two

features. Added *add* as a synonym of *insert*.
12. Introduced *adjust_capacity* which replaces *resize* with a more explicit meaning.
13. Introduced *first* and *last* to get the first and the last character of a STRING.
14. New STRING class invariant: *capacity* >= *count.*
15. Added post-conditions to the *append_** features.
16. Added *extend* and *add_last* as a synonym of *append_character* and *append* as a synonym of *append_string*.
17. Added *is_boolean*, *is_integer*, *is_real* and *is_double* features.
18. Added *same_as* which does a case insensitive comparison between two strings.
19. Added *add_first* and *precede*.
20. Added *infix +* to append two strings.

ARRAY (Note: this list may be incomplete. initial_array.html is the actual reference.)

1. Removed *entry* and *enter*.
2. *item*, *put* and @ are not frozen anymore.
3. Changed the assertions of *make*.
4. Added a precondition to *make_from_array*.
5. Added *capacity* and the class invariant *capacity* >= *count*.
6. Changed the definition of *resize*.
7. Use of *equal* as comparison operator for the new search algorithms (*has*, *index_of*, *reverse_index_of*)
8. Removed ambiguity on *is_equal* and *copy*
9. Renamed *to_c* into *to_external*.
10. Added *is_empty*, *occurrences*, *first* and *last*.
11. Added *add*, *add_first* and *add_last*.
12. Added *remove*, *remove_first* and *remove_last*.

Off standard for the time being

MANIFEST_STRING or CONSTANT_STRING

Very important language/libraries issues were also discussed about the semantics of STRING. In a nutshell, there are two problems:

1. The current semantics of an instruction such as
   foo := "FOO";
   is not very clear and various compilers have different implementation for this. Some create a new object, other consider this instruction as a call to a once function (like for the declaration foo: STRING is "FOO";). This at least must be clarified.
2. STRINGs are mutable. Adding immutable (constant) strings would be a significant improvement, both in terms of performance (potential optimizations) and expressiveness (and safety).

After discussing this, it seem that the correct implementation in the long term would be the following:

1. Mutable STRINGs and immutable CONSTANT_STRING (or another name).
2. "FOO" is the notation for a CONSTANT_STRING.
3. It must be possible to compare two CONSTANT_STRINGs with '=' with the same semantics as is_equal.

NATIVE_ARRAY

Aside the high-level, flexible, ARRAY class, it seems important to provide a standard low-level, C-like, array for optimal performances, like the NATIVE_ARRAY[E] class provided with SmallEiffel.

Back to the top

People Behind

Dominque Colnet (colnet@loria.fr)

Emmanuel Stapf (manus@eiffel.com)

Olivier Zendra (zendra@loria.fr)

Back to the top

Last Update: 20 August 1999.

5.12 Class ARRAY (initial ELKS2000 proposal)

[Please don't report broken URLs because this page is a draft.]

```
indexing

    description: "Resizable 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.)
        require
            valid_indices: (minindex <= maxindex) or (maxindex = minindex - 1)
        ensure
            lower_set: lower = minindex
            upper_set: upper = maxindex
            -- array_set: for i in minindex..maxindex, item(i) = default item

    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.)
       require
            valid_array: a /= Void

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.)
        require
            valid_indices: (minindex <= maxindex) or (maxindex = minindex - 1)
        ensure
            lower_set: lower = minindex
            upper_set: upper = maxindex
            -- array_set: for i in minindex..maxindex, item(i) = default item

    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.)
       require
            valid_array: a /= Void

feature -- Access

    first: like item
            -- First element
        require
            not_empty: not is_empty
        ensure
            definition: Result = item (lower)

    has (v: like item): BOOLEAN
            -- Does Current contain v?
            -- Use equal as comparison operator.
        ensure
            definition: Result = valid_index(index_of(c));
```

```
    index_of (v: like item): INTEGER
            -- Index of first occurrence of v
            -- Use equal as comparison operator.
            -- Upper + 1 if not found
        ensure
            not_found: (Result = upper + 1) = not has(v);
            found: has(v) implies equal(v,item(Result));

    infix "@", item (i: INTEGER): G
            -- Entry at index i
        require
            good_key: valid_index (i)

    last: like item
            -- Last element
        require
            not_empty: not is_empty
        ensure
            definition: Result = item (upper)

    reverse_index_of (v: like item): INTEGER
            -- Index of last occurrence of v?
            -- Use equal as comparison operator.
            -- Lower - 1 if not found
        ensure
            not_found: (Result = upper + 1) = not has(v);
            found: has(v) implies equal(v,item(Result));

feature -- Measurement

    capacity: INTEGER
            -- Current number of elements that Current can contain.

    count: INTEGER
            -- Number of valid indices
        ensure
            definition: Result = upper - lower + 1;

    lower: INTEGER
            -- Minimum valid index

    occurrences (v: like item): INTEGER
            -- Number of times v appears in Current.
            -- Use equal as comparison operator.
        ensure
            non_negative_occurrences: Result >= 0

    upper: INTEGER
            -- Maximum valid index

feature -- Comparison

    is_equal (other: like Current): BOOLEAN
            -- Do both arrays have same lower, upper and
            -- items (compared with is_equal)?
            -- (Redefined from GENERAL.)

feature -- Status report

    is_empty: BOOLEAN
            -- Is array empty?
        ensure
            definition: Result = (count = 0)

    valid_index (i: INTEGER): BOOLEAN
            -- Is i within the bounds of the array?
        ensure
            definition: lower <= i and i <= upper

feature -- Element change

    add (v: like item; i: INTEGER)
        -- Insert v at index i, shifting elements between ranks i and upper rightwards.
       require
```

```
            valid_insertion_index: lower <= i and i <= upper + 1
        ensure
            inserted: item (i) = v;
            same_lower: lower = old lower;
            higher_upper: upper = old upper + 1

    add_first (v: like item)

            -- Insert v at index lower, shifting all elements rightwards.
        ensure
            inserted: first = v;
            same_lower: lower = old lower;
            higher_upper: upper = old upper + 1

    add_last (v: like item)
            -- Append v at end.
        ensure
            inserted: last = v;
            same_lower: lower = old lower;
            higher_upper: upper = old upper + 1

    force (v: like item; i: INTEGER)
            -- Assign v to i-th entry.
            -- Always applicable: resize the array if i falls out of
            -- currently defined bounds; preserve existing items.
            -- New entries if any are initialized to default value.
        ensure
            put: item (i) = v;
            higher_count: count >= old count

    put (v: like item; i: INTEGER)
            -- Assign v to i-th entry.
        require
            good_index: valid_index (i)
        ensure
            put: item (i) = v

feature -- Removal

    remove (i: INTEGER)
            -- Remove element at index i and shift elements between ranks i+1 and upper lef
        require
            good_index: valid_index (i)
        ensure
            same_lower: lower = old lower;
            smaller_upper: upper = old upper - 1

    remove_first
            -- Remove first element and shift all elements leftwards.
        require
            not_is_empty: not is_empty
        ensure
            same_lower: lower = old lower;
            smaller_upper: upper = old upper - 1

    remove_last
            -- Remove last element.
        require
            not_is_empty: not is_empty
        ensure
            same_lower: lower = old lower;
            smaller_upper: upper = old upper - 1

    wipe_out
            -- Remove all elements.
        ensure
            is_empty: is_empty
            same_capacity: capacity = old capacity

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 in the
            -- intersection between [old lower .. old upper] and
            -- [minindex .. maxindex].
            -- New positions are initialized to their default value.
        require
            valid_indices: (minindex <= maxindex) or (maxindex = minindex - 1)
        ensure
            lower_set: lower = minindex
            upper_set: upper = maxindex

feature -- Conversion

    to_external: POINTER
            -- Address of the storage area containing the actual sequence of elements,
            -- to be passed to some external (non-Eiffel) routine.
            -- Keep in mind that this storage area is subject to garbage collection.

feature -- Duplication

    copy (other: like Current)
            -- Reinitialize by copying all the items of other.
            -- Shallow copy: if the elements are references, only the references
            -- are copied, not the object they point to.
            -- (Redefined from GENERAL.)
        ensure
            same_lower: lower = other.lower;
            same_upper: upper = other.upper;
            -- same_content: For every i in lower..upper,
            --     item (i) = other.item (i)

invariant

    consistent_count: count = upper - lower + 1;
    non_negative_count: count >= 0;
    good_capacity: capacity >= count

end
```

Last Updated: 20 August 1999

5.13 Class STRING (initial ELKS2000 proposal)

[Please don't report broken URLs because this page is a draft.]

```
indexing

    description: "Resizable sequences of characters, accessible through %
        %integer indices in a contiguous range. If not empty, first element%
        %is at index 1."

class interface

    STRING

creation

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

    make_from_external (ptr: POINTER)
            -- Set Current with a copy of the content of ptr.
            -- Assume ptr is a null-terminated memory area containing
            -- only characters.
            -- The extra null character is not part of the Eiffel string
        require
            ptr_exists: ptr /= default_pointer

    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
        ensure
            count_set: count = s.count

feature -- Initialization

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

    make_from_external (ptr: POINTER)
            -- Set Current with a copy of the content of ptr.
            -- Assume ptr is a null-terminated memory area containing
            -- only characters.
            -- The extra null character is not part of the Eiffel string
        require
            ptr_exists: ptr /= default_pointer

    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
        ensure
            count_set: count = s.count

feature -- Access
```

```
first: CHARACTER
        -- First character of Current.
    require
        not_empty_string: not is_empty
    ensure
        good_result: Result = item (1)

has (c: CHARACTER): BOOLEAN
        -- Does Current contain c ?
    ensure
        definition: Result = (index_of(c) /= 0);

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

index_from (c: CHARACTER; start_index: INTEGER): INTEGER
        -- Index of first occurrence of c at or after start_index;
        -- 0 if none.
    require
        valid_start_index: valid_index (start_index)
    ensure
        non_negative_result: Result >= 0;
        valid_result: Result > 0 implies valid_index (Result)
        result_after_start_index: Result > 0 implies Result >= start_index;
        at_this_index: Result > 0 implies item (Result) = c;
        -- none_before: For every i in start_index..Result, item (i) /= c
        -- zero_iff_absent:
        --     (Result = 0) = For every i in start_index..count, item (i) /= c

index_of (c: CHARACTER): INTEGER
        -- Index of first occurrence of c;
        -- 0 if none.
    ensure
        empty_definition: empty implies Result = 0;
        non_empty_definition: not empty implies Result = index_from (c, 1);

infix "@", item (i: INTEGER): CHARACTER
        -- Character at index i
    require
        good_key: valid_index (i)

last: CHARACTER
        -- Last character of Current.
    require
        not_empty_string: not is_empty
    ensure
        good_result: Result = item (count)

reverse_index_from (c: CHARACTER; last_index: INTEGER): INTEGER
        -- Index of first occurrence of c at or before last_index;
        -- 0 if none.
    require
        valid_last_index: valid_index (last_index)
    ensure
        non_negative_result: Result >= 0;
        valid_result: Result > 0 implies valid_index(Result)
        result_before_last_index: Result <= last_index;
        at_this_index: Result > 0 implies item (Result) = c;
        -- none_before: For every i in Result..last_index, item (i) /= c
        -- zero_iff_absent:
        --     (Result = 0) = For every i in 1..last_index, item (i) /= c

reverse_index_of (c: CHARACTER): INTEGER
        -- Index of last occurrence of c;
        -- 0 if none.
    ensure
        empty_definition: empty implies Result = 0;
        non_empty_definition: empty implies Result = reverse_index_from (c, count);

reverse_string_index_from (other: STRING; last_index: INTEGER): INTEGER
```

```
            -- Index of first occurrence of other at or before last_index;
            -- 0 if none.
        require
            other_not_empty: not other.empty;
            valid_last_index: valid_index (last_index)
        ensure
            non_negative_result: Result >= 0;
            valid_result: Result > 0 implies valid_index (Result)
            result_before_last_index: Result > 0 implies Result <= last_index;
            at_this_index: Result > 0 implies other.is_equal (substring (Result, Result - 1
            -- none_before:
            --     For every i in Result..last_index, not other.is_equal (substring (i, i -
            -- zero_iff_absent:
            --     (Result = 0) = For every i in 1..last_index, not other.is_equal (substri

    reverse_string_index_of (other: STRING): INTEGER
            -- Index of last occurrence of other;
            -- 0 if none.
        require
            other_not_empty: not other.empty
        ensure
            empty_definition: empty implies Result = 0;
            non_empty_definition: not empty implies Result = reverse_string_index_from (oth

    string_index_from (other: STRING; start_index: INTEGER): INTEGER
            -- Index of first occurrence of other at or after start;
            -- 0 if none.
        require
            other_not_empty: not other.empty;
            valid_start_index: valid_index (start_index)
        ensure
            non_negative_result: Result >= 0;
            valid_result: Result > 0 implies valid_index (Result)
            result_after_start_index: Result > 0 implies Result >= start_index;
            at_this_index: Result > 0 implies other.is_equal (substring (Result, Result - 1
            -- none_before:
            --     For every i in start_index..Result, not other.is_equal (substring (i, i
            -- zero_iff_absent:
            --     (Result = 0) = For every i in start_index..count, not other.is_equal (su

    string_index_of (other: STRING): INTEGER
            -- Index of first occurrence of other;
            -- 0 if none.
        require
            other_not_empty: not other.empty
        ensure
            empty_definition: empty implies Result = 0;
            non_empty_definition: not empty implies Result = reverse_string_index_from (oth

feature -- Measurement

    count: INTEGER
            -- Current number of characters making up the string

    capacity: INTEGER
            -- Current number of characters that the string
            -- can contain.

    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

    infix "<" (other: like Current): 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)

    same_as (other: STRING): BOOLEAN
            -- Is string made of same character sequence as other?
            -- Case insensitive comparison.
        require
            other_exists: other /= Void

    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

    is_empty: BOOLEAN
            -- Is string empty?
        ensure
            definition: Result = (count = 0)
```

```
    valid_index (index: INTEGER): BOOLEAN
            -- Is i within the bounds of the string?
        ensure
            definition: Result = (1 <= index and index <= count)

    is_boolean: BOOLEAN
            -- Is Current representing a boolean ("true" or "false" with no
            -- case consideration)?

    is_double: BOOLEAN
            -- Can Current be read as a DOUBLE?

    is_integer: BOOLEAN
            -- Can Current be read as an INTEGER?

    is_real: BOOLEAN
            -- Can Current be read as a REAL?

feature -- Element change

    add, insert (c: CHARACTER; i: INTEGER)
            -- Insert c at index i, shifting characters between ranks i and count rightwards
        require
            valid_insertion_index: 1 <= i and i <= count + 1
        ensure
             inserted: item (i) = c
             one_more_character: count = old count + 1

    add_first, precede (c: CHARACTER)
            -- Prepend c at beginning.
        ensure
            inserted: first = c
            one_more_character: count = old count + 1

    add_last, append_character, extend (c: CHARACTER)
            -- Append c at end.
        ensure
            inserted: last = c
            one_more_character: count = old count + 1

    append, append_string (s: STRING)
            -- Append a copy of s at end.
        require
            other_exists: s /= Void
        ensure
            new_count: count = old count + s.count
            -- appended: For every i in 1.. s.count,
            --     item (old count + i) = s.item (i)

    append_boolean (b: BOOLEAN)
            -- Append the string representation of b at end.
        ensure
            boolean_inserted: reverse_index_of (b.out) = count - b.out.count
            new_count: count = old count + b.out.count + 1

    append_double (d: DOUBLE)
            -- Append the string representation of d at end.
        ensure
            double_inserted: reverse_index_of (d.out) = count - d.out.count
            new_count: count = old count + d.out.count + 1

    append_integer (i: INTEGER)
            -- Append the string representation of i at end.
        ensure
            integer_inserted: reverse_index_of (i.out) = count - i.out.count
            new_count: count = old count + i.out.count + 1

    append_real (r: REAL)
            -- Append the string representation of r at end.
        ensure
            real_inserted: reverse_index_of (r.out) = count - r.out.count
            new_count: count = old count + r.out.count + 1

    fill (c: CHARACTER)
```

```
            -- Replace every character with c.
        ensure
            same_count: old count = count
            filled: occurrences(c) = count

    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_string (s: like Current; i: INTEGER)
            -- Insert s at index i, shifting characters between ranks i and count rightwards
        require
            string_exists: s /= Void;
            valid_insertion_index: 1 <= i and i <= count + 1
        ensure
            new_count: count = old count + s.count
            inserted: s.is_equal (substring (i, i - 1 + s.count))

    left_adjust
            -- Remove leading white space.
        ensure
            shorter: count <= old count
            first_not_white_space: not is_empty implies ("%T%R%N ").index_of (first) = 0

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

    right_adjust
            -- Remove trailing white space.
        ensure
            shorter: count <= old count
            last_not_white_space: not is_empty implies ("%T%R%N ").index_of (last) = 0

    subcopy (other: like Current; start_pos, end_pos, from_index: INTEGER) is
            -- Copy characters of other within bounds start_pos and
            -- end_pos to current string starting at index from_index.
        require
            other_not_void: other /= Void;
            valid_start_pos: other.valid_index (start_pos)
            valid_end_pos: other.valid_index (end_pos)
            valid_bounds: start_pos <= end_pos
            valid_from_index: valid_index (from_index)
            enough_space: valid_index (from_index + end_pos - start_pos)
        ensure
            same_count: old count = count
            copied: substring (from_index, from_index + end_pos - start_pos).is_equal
                    (other.substring (start_pos, end_pos))

    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, shifting characters between ranks i + 1 and count le
        require
            valid_removal_index: valid_index (i)
        ensure
            new_count: count = old count - 1
```

```
    wipe_out
            -- Remove all characters.
        ensure
            empty_string: is_empty
            same_capacity: capacity = old capacity

feature -- Resizing

    adjust_capacity (n: INTEGER)
            -- Rearrange string so that it can accommodate
            -- at least n characters.
            -- Do not lose any previously entered character.
        require
            no_item_lost: n >= count
        ensure
            enough_capacity: capacity >= n

feature -- Conversion

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

    to_double: DOUBLE
            -- Double value; for example, when applied to "123.0",
            -- will yield 123.0 (double)
        require
            is_double: is_integer or is_real or is_double

    to_external: POINTER
            -- Address of the storage area containing the actual sequence of characters,
            -- to be passed to some external (non-Eiffel) routine.
            -- Keep in mind that this storage area is subject to garbage collection.

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

    to_lower
            -- Convert to lower case.

    to_real: REAL
            -- Real value;
            -- for example, when applied to "123.0", will yield 123.0
        require
            is_real: is_integer or is_real

    to_upper
            -- Convert to upper case.

feature -- Duplication

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

    infix "+" (other: STRING): like Current
            -- Create a new object which is the concatenation of Current and other.
        require
            other_exists: other /= Void
        ensure
            result_count: Result.count = count + other.count

    substring (start_index, end_index: INTEGER): like Current
```

```
            -- Create a substring containing all characters from start_index
            -- to end_pos included.
        require
            valid_start_index: valid_index (start_index)
            valid_end_index: valid_index (end_index)
            meaningful_interval: start_index <= end_index
        ensure
            new_result_count: Result.count = end_index - start_index + 1
            -- original_characters: For every i in 1..end_index - start_index + 1,
            --     Result.item (i) = item (start_index + i -1)

feature -- Output

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

invariant

    non_negative_count: count >= 0
    capacity_big_enough: capacity >= count

end
```

Last Updated: 20 August 1999

ELKS 2000 proposal changelog

Preamble

This page lists the changes made to the initial ELKS2000 proposal (initial_string.html and initial_array.html) to obtain the current ELKS2000 proposal (current_string.html and current_array.html).

These changes result from the discussions in the NICE mailing list dedicated to library issues, http://www.egroups.com/group/eiffel-nice-library.

Of course, only those changes that have been agreed upon are put in here.

Changes (latest first)

15 September 1999:
- Fixed postcondition of `append, append_string` and `insert_string` in STRING.

01 September 1999:
- Moved `adjust_capacity` from `Resizing` feature clause to `Optimization` feature clause in ARRAY and STRING.
- Added `reindex` feature in class ARRAY.

30 August 1999:
- Updated comments for `capacity` in ARRAY and STRING.
- Made postcondition `same_lower` explicit in `wipe_out` of ARRAY and STRING.

20 August 1999:
- Added these ELKS2000 pages.
- Added `adjust_capacity` in ARRAY (was mistakenly forgotten in the initial proposal), like in STRING.
- Fixed mistakes in postcondition `--none_before:` for routines `index_from`, `string_index_from`, `reverse_index_from`, and `reverse_string_index_from` of class STRING.

Back to the top

Maintainers: Dominique Colnet.

Last Update: 15 September 1999.