12. Input and output (I/O) of intervals

12.1. Overview. This standard specifies conversion from a text string that holds an interval literal to an interval internal to a program (input), and the reverse (output). The method by which strings are read from, or written to, a character stream is language- or implementation-defined.

Containment shall hold on input and output so that, when a program computes an enclosure of some quantity given an enclosure of the data, a correctly written program can ensure this holds all the way from text data to text results.

In addition to normal I/O, the standard requires each interval type $\mathbb{T}$ to have an *exact representation*. This has two parts: operations to convert any internal $\mathbb{T}$-interval $\boldsymbol{x}$ to a string $\boldsymbol{s}$, and back again to recover $\boldsymbol{x}$ exactly; and documentation of the mathematical relation between $\boldsymbol{s}$ and the Level 1 interval value of $\boldsymbol{x}$.

12.2. Input. Input is provided for each supported bare or decorated interval type $\mathbb{T}$ by the $\mathbb{T}$-version of `text2interval`($\boldsymbol{s}$), where $\boldsymbol{s}$ is a string, as specified in §11.11.8. It accepts an arbitrary interval literal $\boldsymbol{s}$ and returns a $\mathbb{T}$-interval enclosing the Level 1 value of $\boldsymbol{s}$.
[*Note. This provides the basis for free-format input of interval literals from a text stream, as might be provided by overloading the* >> *operator in C++.*]

12.3. Output. An implementation shall provide an operation

`interval2text`($\boldsymbol{X}$, cs)

where cs is optional. $\boldsymbol{X}$ is a bare or decorated interval datum of any supported interval type $\mathbb{T}$, and cs is a string, the conversion specifier. The operation converts $\boldsymbol{X}$ to a valid interval literal string $\boldsymbol{s}$, see§11.11.1, which shall be related to $\boldsymbol{X}$ as follows, where $\boldsymbol{Y}$ is the Level 1 value of $\boldsymbol{s}$.

(i) Let $\mathbb{T}$ be a bare type. Then $\boldsymbol{Y}$ shall contain $\boldsymbol{X}$, and shall be empty if $\boldsymbol{X}$ is empty.
(ii) Let $\mathbb{T}$ be a decorated type. If $\boldsymbol{X}$ is NaI then $\boldsymbol{Y}$ shall be NaI. Otherwise, write $\boldsymbol{X} = \boldsymbol{x}_{dx}$, $\boldsymbol{Y} = \boldsymbol{y}_{dy}$. Then
- $\boldsymbol{y}$ shall contain $\boldsymbol{x}$, and shall be empty if $\boldsymbol{x}$ is empty.
- dy shall equal dx, except in the case that $dx =$ `com` and overflow occurred, that is, $\boldsymbol{x}$ is bounded and $\boldsymbol{y}$ is unbounded. Then dy shall equal `dac`.

[*Note.* $\boldsymbol{Y}$ *being a Level 1 value is significant. E.g., for a bare type* $\mathbb{T}$*, it is not allowed to convert* $\boldsymbol{X} = \emptyset$ *to the string* `garbage`*, even though converting* `garbage` *back to a bare interval at Level 2 by* $\mathbb{T}$`-text2interval` *gives* $\emptyset$*, because* `garbage` *has no Level 1 value as a bare interval literal.*]

The tightness of enclosure of $\boldsymbol{X}$ by the value $\boldsymbol{Y}$ of $\boldsymbol{s}$ is language- or implementation-defined.

If present, cs controls the layout of the string $\boldsymbol{s}$ and its syntactic parts, see§11.11.1, for instance l, u, m and r where relevant. The standard does not specify cs but among the user-controllable elements should be the following:

(i) The overall field width (the length of $\boldsymbol{s}$), and whether output is in inf-sup or uncertain form.
(ii) How Empty, Entire and NaI are output, e.g., whether lower or upper case, and whether Entire becomes `[Entire]` or `[-Inf, Inf]`.
(iii) For l, u, m and r, the field width and the number of places after the point or the number of significant figures. There should be a choice of radix.

If cs is absent, output should be in a general-purpose layout (analogous, e.g., to the `%g` specifier of `fprintf` in C). There should be a value of cs that selects this layout explicitly.
[*Note. This provides the basis for free-format output of intervals to a text stream, as might be provided by overloading the* << *operator in C++.*]

If $\mathbb{T}$ is a 754-conforming bare type, there shall be a value of cs that produces behavior identical with that of `interval2exact`, below. That is, the output is an interval literal that, when read back by `text2interval`, recovers the original datum exactly.

12.4. Exact representation. For any supported bare interval type $\mathbb{T}$ an implementation shall provide operations `interval2exact` and `exact2interval`. Their purpose is to provide a portable text representation of every bare interval datum. They are characterized by the **recovery requirement**:

> For any $\mathbb{T}$-datum $\boldsymbol{x}$, the value $\boldsymbol{s} =$ `interval2exact`($\boldsymbol{x}$) is a string, such that $\boldsymbol{y} =$ `exact2interval`($\boldsymbol{s}$) is a $\mathbb{T}$-datum equal to $\boldsymbol{x}$.

[*Note. This is equality at Level 2.* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *may differ at Level 3, e.g., a zero endpoint might be stored as* -0 *in one and* $+0$ *in the other.*]

If $\mathbb{T}$ is a 754-conforming type, the string $\boldsymbol{s}$ shall be an interval literal. The operation `exact2interval` shall be the same as `text2interval` for this type. If $\boldsymbol{x}$ is nonempty, $\boldsymbol{s}$ shall be of inf-sup form $[l, u]$ with l and u represented exactly. If $\mathbb{T}$ is a binary type, l and u if finite shall be represented in the hexadecimal-significand form of 754§5.12.3. If $\mathbb{T}$ is a decimal type, they shall be represented as decimal numbers.

If $\mathbb{T}$ is not 754-conforming, there are no restrictions on the form of the string $\boldsymbol{s}$ apart from the above recovery requirement. However, the algorithm by which `interval2exact` converts $\boldsymbol{x}$ to $\boldsymbol{s}$ is regarded as part of the definition of the type and shall be documented by the implementation.

For any type $\mathbb{T}$, since `exact2interval` inputs a string $\boldsymbol{s}$ and outputs a bare interval, it is a constructor similar to `text2interval`, having a set V of *valid* strings that it regards as denoting an interval. This set includes by definition the set V_0 of all strings that can be output by `interval2exact`. For a 754-conforming type, V is the set of valid interval literals, and is larger than V_0 since any such literal can be written in many equivalent ways. For other types $\mathbb{T}$, the documentation shall specify the set V, which might or might not be strictly larger than V_0.

As with `text2interval` (see §11.11.8), `exact2interval` shall return $\emptyset$ and raise a language- or implementation-defined exception when its input is invalid.