Is a given input a valid TEX ⟨*dimen*⟩?

Udo Wermuth

Abstract

This article discusses the question of how one can determine if a given string of characters represents a valid dimension for TEX. A macro that looks and behaves like a Boolean conditional is implemented to answer the question.

1 Introduction

This text is a follow-on article to [3], which explains how one can decide if a given string of characters is a valid number for TEX; the macro implemented there is named `\ifisint`. In the current article we look at the problem to decide if a given input is a valid dimension for TEX.

This paper also explains the implementation of a macro named `\ifisdim` with the structure known from `\ifisint`. It is essential that a reader of this article has studied [3] as this text often refers to [3] without repeating the introduced techniques.

Contents. This article follows the analysis found in [3] and describes how to implement a TEX macro looking like a Boolean conditional to answer the question of the title; as mentioned above, the macro is named `\ifisdim`. The expectations formulated in [3], section 2, apply to `\ifisdim` accordingly.

Section 2 lists facts about TEX's dimensions that are important to understand `\ifisdim`. Section 3 contains the code for `\ifisdim`.

2 About dimensions

It's too naïve to say that a dimension is a TEX number and a unit; what's correct is that a dimension consists of a *numeric part* and a unit ([1], pp. 270–271). One option for the numeric part is a TEX ⟨*number*⟩, i.e., an integer. All encodings (see [3]) are allowed but not their full range; see below. Another option is the ⟨*decimal constant*⟩, i.e., a number followed by a period or comma and a sequence of digits that builds the fraction. TEX reads all digits that it finds after the period or comma but at most the first seventeen can influence the value of the dimension; see §452 of [2].

TEX respects different traditions of writing decimal constants and therefore accepts two symbols as the separator between the integer part and the fraction. TEX also respects the history of different printing traditions and comes with plenty of units. There are nine ⟨*physical unit*⟩s ([1], p. 57): One can use `pt` (point) and `pc` (pica) from the American stan-

doi.org/10.47397/tb/45-1/tb139wermuth-isdim

dardization in the 19th century or `dd` (didot point) and `cc` (cicero) based on the practice of François-Ambroise Didot in the 18th century. Next, TEX accepts `in` (inch) or units in the metric system: `mm` (millimeter) and `cm` (centimeter). It introduced `bp` (big point) and `sp` (scaled point). Moreover, TEX also knows about traditional units used by typesetters, `ex` (x-height) and `em` (quad width), that depend on the font that's currently in use ([1], p. 60).

The units `ex` and `em` don't belong to the physical units as another parameter is required to determine their values: a font. Here we fix the font to TEX's default font `cmr10` and include both units in the tests of the new macro `\ifisdim`.

The two syntactic quantities ⟨*mudimen*⟩ and ⟨*fil dimen*⟩ carry the word "dimen" in their description but they cannot be assigned to a `\dimen` register. ⟨*mudimen*⟩ ([1], p. 270) must be used with a muskip, which is a glue specification. ⟨*fil dimen*⟩ ([1], p. 271) only occurs in stretch or shrink components of skips and muskips; again it's part of glue specifications. `\ifisdim` doesn't recognize these quantities as valid TEX dimensions.

A valid unit is either one of the nine ⟨*physical unit*⟩s that can be preceded by the keyword `true` to protect it against magnification or the two font-dependent units `em` and `ex`. All units are keywords so that they can be written with category 11 or 12 characters, in upper-, lower-, or mixed-case, and with optional spaces in front of them; see [1], p. 268.

Dimensions are internally represented by TEX in scaled points and TEX uses the unit `pt` if it has to show a stored one. The numeric part of a dimension in scaled points must lie between $-2^{30}+1$ and $2^{30}-1$. Thus, the range is smaller than the one for numbers; see [3]. 1 sp is a very small distance, 65536 sp give 1 pt and that means the maximum decimal constant for the unit `pt` is much smaller than $2^{30} - 1$.

Table 1: Ranges for physical units

unit	**max. decimal constant**[†]	**shown as**
pt	16383.99999237060546874	16383.99998pt[*]
pc	1365.33333587646484374	16383.99994pt
in	226.70540618896484374	16383.99915pt
bp	16322.78954315185546874	16383.99998pt[*]
dd	15312.02584075927734374	16383.99997pt
cc	1276.00215911865234374	16383.99995pt
mm	5758.31742095947265624	16383.99997pt
cm	575.83174896240234374	16383.99997pt
sp	1073741823.99999999999999999	16383.99998pt[*]
Using the `\fontdimen` of `cmr10`:		
ex	3805.32811737060546874	16383.99997pt
em	1638.39749908447265624	16383.99991pt

[†] With the (at most) seventeen significant decimal places.

[*] TEX represents this value as $2^{30} - 1\,\mathrm{sp} = 1073741823\,\mathrm{sp}$.

The line in Table 1 for the unit `pt` tells us that an infinite number of input strings with this unit are mapped to TEX's largest dimension. Plain TEX sets `\maxdimen` to 16383.99999 pt but TEX shows it as 16383.99998 pt. When TEX has to show a dimension it outputs at most five digits ([2], §103).

Enter the values 16383.99997711181640625 pt, 16322.78952789306640625 bp, and 1073741823 sp to specify `\maxdimen` with the smallest decimal constants for the three units that can do that.

3 The code for `\ifisdim`

A valid dimension is (1) an integer followed by a valid unit or (2) a ⟨*decimal constant*⟩ with a valid unit as described in section 2. Thus, we encounter the three error messages of TEX when it reads an integer as discussed in [3]. The *scan_dimen* procedure in [2], part 26, adds a few new error situations. Sections 456 and 459 contain the message "Illegal unit of measure" once for dimensions and once for ⟨*mudimen*⟩. And section 460 includes the error message "Dimension too large". In total we have to deal with five error messages that TEX might show when it reads a dimension.

The first new error message means that TEX has found (or inserted) a numeric part and expects now one of the valid units — maybe prefixed with the keyword `true`. If it doesn't find one it inserts the unit `pt` to get a valid dimension. The numeric part might have been generated by TEX if it wasn't able to read a number, i.e., TEX might have inserted a zero as described in [3].

The second error message tells us that the combination of numeric part and unit results in a scaled-point value larger than 1073741823 sp. The help text of the error message informs us that TEX throws the input away and uses its largest dimension instead.

Analysis. Let's list all possible scenarios. Several of the following cases appear with and without more input. We know how to handle this from `\ifisint` so it isn't mentioned here again. Only if it is important that no more data is available is it handled as a separate case.

1. TEX reads a valid dimension.
2. TEX doesn't find a numeric part, uses 0 instead, finds a valid unit.
3. TEX doesn't find a numeric part, uses 0 instead, doesn't find a unit, uses `pt` instead.
4. TEX doesn't find a numeric part, uses 0 instead, finds an invalid unit, uses `pt` instead.
5. TEX finds as numeric part a number larger than 2147483647, uses 2147483647 instead, finds a

valid unit, thus the dimension is too large and TeX uses `\maxdimen` instead.

6. TeX finds as numeric part a number larger than 2147483647, uses 2147483647 instead, finds no valid unit, inserts `pt`, thus the dimension is too large and TeX uses `\maxdimen` instead.
7. TeX finds a numeric part and a valid unit but the combination creates a dimension that's too large, uses `\maxdimen` instead.
8. TeX finds a numeric part but no unit, inserts unit `pt` and builds a valid dimension.
9. TeX finds a numeric part but no unit, inserts unit `pt`, the combination creates a dimension that's too large, uses `\maxdimen` instead.

The list is much longer than the one in [3] for `\ifisint`. But a second check shows that several cases can be deleted. Cases 5 and 6 are just special cases of 7 with more error messages. Next, cases 3, 8 and 9 can be avoided if the sentinel (see [3]) is a valid unit, for example, `mm`. This gives a width for an hbox with an assignment [3] that disagrees with the width of the string '`mm`'. And this happens with case 4 too as the invalid unit and the sentinel remain.

We are left with cases 2 and 7 for invalid dimensions. Looking at [3] we are faced in essence with the same cases but this time each case involves units. For example, case 2 excludes input like "`'pt`" that TeX transforms into "`0pt`". So it looks like we have to execute the three tests of [3] together with all valid units. But no, it doesn't hurt to exclude input with invalid units. All we have to do is to check that the input has at most three characters (without the keyword `true`) and starts with '`'`' or '`"`'. The incomplete alphabetic constant is again moved; here it destroys the unit and generates an error.

Case 7 remains. The solution in [3] was to use a list of the canonical forms of the largest integer in all encodings. So here we need a list of the canonical forms for `\maxdimen`. But there seems to be no simple form for the infinitely many input strings that represent `\maxdimen`, as we saw in Table 1.

In order to distinguish case 7 from case 1 we need to do some calculations: We need to determine the input value in scaled points and compare the result against $2^{30}-1\,\mathrm{sp}$. To do that without risk of getting a false result we use three elements.

a. The integer part of the numeric part: `\II@int`.
b. The fractional part plus the unit: `\II@frac`.
c. The unit, maybe prefixed with `true`: `\II@unit`.

The key to success is the fact that in TeX the range for numbers is larger than the range for dimensions expressed in scaled points. Thus the following computation doesn't generate a "Dimension too large" error if `\II@int` is at most as large as the integer part of the maximum decimal constant for `\II@unit` according to Table 1.

```
\dimen255=\II@int\II@unit
\count255=\dimen255 % coerce dimension to number
\advance\count255 by \II@frac
\def\II@calc{\number\count255 }
```

`\II@calc` contains the sum of the number of scaled points of `\dimen255` and `\II@frac`; see [1], p. 270.

How do we get the required information? If we have a dimension, `\II@dist=\II@int\II@frac`, two assignments fill the variables, with an error message if `\II@dist` contains neither a decimal point nor a decimal comma.

```
\afterassignment\II@frac \II@int=\II@dist
```

It's easy to avoid the error by inserting a zero.

It is not much harder to identify the unit. We assign the digits of the fraction — after removing the period or comma — to a `\count` register, leaving the two characters of the unit. Using an hbox we can distinguish the three strings '`pt`', '`bp`', and '`sp`' by their widths. (In general it is not possible to identify all units, for example, the strings '`bp`' and '`dd`' have the same width. But we are only interested in the width if the dimension equals `\maxdimen` and that cannot happen with '`dd`'; see Table 1.) The keyword `true` must also be considered; its width is subtracted if the width of the hbox exceeds a certain value.

There is one problem: Keywords can be written in different ways with lower- and uppercase characters; the characters might even be of category 12. We need to transform them into a canonical form, for example, lowercase letters, to get a unique width.

Thus we need to realize the following procedure.

Step 1: 1) Remove signs; add sentinel. 2) Test that case 2 is excluded. 3) Otherwise return false.

Step 2: Get the parts: 1) `\II@int`, 2) `\II@frac`, and 3) `\II@unit` (as a width).

Step 3: 1) Assign the input to a `\dimen` register inside an hbox. 2) Test that the box width is the width of the sentinel; 3) otherwise return false (includes cases with more data).

Step 4: 1) Return true if the dimension isn't TeX's `\maxdimen`. 2) Otherwise test if `\II@calc` is TeX's `\maxdimen`. 3) If no, return false (case 7). 4) Otherwise return true (case 1).

This procedure works with a lot of intentional errors that TeX reports while `\batchmode` is active. Thus TeX's limit of 100 error messages per paragraph ([2], §76) is reached much earlier than with `\ifisint`.

My implementation. The following private control words — the two declarations `\ifII@itis` and

\II@font, the macros \II@W, \II@octW, \II@hexW, \II@rmsign, and \II@endrm, and the \let-assignments \Boolend and \IIcurrentmode — are reused from the code of [3]. Their code is marked with two comment characters at the right end of the lines in the following code. You might delete this code if you load via \input the file that contains the code of [3].

```
\catcode‘\@=11
\newif\ifII@itis      %% reused from \ifisint %%
\def\II@rmsign #1{\ifx#1+\else\ifx#1-\else    %%
 \II@endrm#1\fi\fi\II@rmsign}%  remove signs: %%
\def\II@endrm #1\fi\fi#2{\fi\fi#1}% ‘+’ & ‘-’ %%
\let\Boolend=\iffalse \font\II@font=cmr10 %   %%
\let\IIcurrentmode=\errorstopmode % CONFIGURE %%
\def\II@W{W}\def\II@octW{’W}\def\II@hexW{"W}% %%
%% declarations
\newdimen\II@frac
\countdef\II@cnt=255 \dimendef\II@dim=255
%% helper macros; some use \ifisint’s sentinel W
\def\II@bad #1#2#3#4#5#6\II@end{%   numeric part
 \def\II@id{#1W}% is missing but maybe with unit
 \edef\II@X{#6}\ifx\II@X\empty
  \edef\II@X{#5}\ifx\II@X\empty\else\II@Bad\fi
 \else \edef\II@X{\II@mklc#2#3#4W}%
  \ifx\II@X\II@rueW
  \else\ifx\II@X\II@truW\II@Bad
  \else \II@itistrue
 \fi\fi\fi}
\def\II@rueW{rueW}\def\II@truW{truW}
\def\II@Bad{\ifx\II@id\II@W
 \else\ifx\II@id\II@octW
 \else\ifx\II@id\II@hexW
 \else \II@itistrue
 \fi\fi\fi}
\def\II@mklc #1{\if#1pp\else\if#1Pp\else
 \if#1tt\else\if#1Tt\else
 \if#1bb\else\if#1Bb\else
 \if#1ss\else\if#1Ss\else
 \if#1rr\else\if#1Rr\else
 \if#1uu\else\if#1Uu\else
 \if#1ee\else\if#1Ee\else
  \II@endlc#1\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
 \fi\fi\fi \II@mklc}%  ‘W’ and ‘m’ stop \II@mklc
\def\II@endlc #1\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
 \fi\fi\fi\fi#2{\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
 \fi\fi\fi\fi#1}
\def\II@getfrac #1mm\II@end{\global\II@frac=0#1}
\def\II@getcalc{%\II@calc=coerced\II@int\II@frac
 \ifdim\II@unit=26.11119pt %  \II@unit is ‘‘pt’’
  \II@dim=\ifnum\II@int<16384
   \II@int\else 0\fi pt
 \else\ifdim\II@unit=27.77786pt %   it is ‘‘bp’’
  \II@dim=\ifnum\II@int<16323
   \II@int\else 0\fi bp
 \else\ifdim\II@unit=26.16673pt %   it is ‘‘sp’’
  \II@dim=\ifnum\II@int<1073741824
   \II@int\else 0\fi sp
 \else \II@dim=0pt \fi\fi\fi
 \II@cnt=\II@dim \advance\II@cnt by \II@frac
 \edef\II@calc{\number\II@cnt}}
\def\II@point #1#2\II@end{% assign digits of the
 \afterassignment\II@mklc  % fraction to \II@cnt
 \ifx#1.\II@cnt=0#2%
 \else\ifx#1,\II@cnt=0#2%
 \else \II@cnt=0#1#2%
 \fi\fi}
\def\II@getunit #1{\afterassignment\II@hdlfrac
 \II@cnt=#1\relax}
\def\II@rmtrue{\ifdim\wd0>40pt \the\II@dim
 \else \the\wd0 \fi}
%% main macro
\def\ifisdim #1\Boolend{\II@itisfalse     % S1.3
 \edef\II@dist{\II@rmsign#1mm}%             S1.1
 \edef\II@dist{\expandafter\II@rmsign\II@dist}%
 \expandafter\II@bad
  \II@dist\empty\empty\empty\empty\II@end % S1.2
 \ifII@itis                          % S4.1, S4.4
  \wlog{=== start ignore}\batchmode\begingroup
   \setbox0=\hbox{\II@font
    \afterassignment\II@getfrac
     \II@cnt=\II@dist\II@end              % S2.2
    \xdef\II@int{\the\II@cnt}}%             S2.1
   \setbox0=\hbox{\II@font
    \afterassignment\II@point
     \II@cnt=\II@dist\II@end}\II@dim=\wd0
   \advance\II@dim by -17.80559pt % width ‘true’
   \xdef\II@unit{\II@rmtrue}%               S2.3
   \setbox0=\hbox{\II@font\II@dim=#1mm%     S3.1
    \xdef\II@val{\ifdim\II@dim<0pt-\fi
     \the\II@dim}}%
   \xdef\II@wd{\the\wd0}%
  \endgroup\IIcurrentmode\wlog{=== stop ignore}%
  \ifdim\II@wd=16.66672pt % width ‘‘mm’’    S3.2
   \ifdim\II@val=\maxdimen \II@getcalc
    \ifnum\II@calc=1073741823 %             S4.2
    \else \II@itisfalse                   % S4.3
   \fi\fi
  \else \II@itisfalse                     % S3.3
 \fi\fi \ifII@itis}
\catcode‘\@=12
```

References

[1] Donald E. Knuth, *The TEXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984.

[2] Donald E. Knuth, *TEX: The Program*, Volume B of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986.

[3] Udo Wermuth, “Is a given input a valid TEX ⟨*number*⟩?”, *TUGboat* **45**:1 (2024), 106–109. tug.org/TUGboat/tb45-1/tb138wermuth-isint.pdf

◇ Udo Wermuth
Dietzenbach, Germany
u dot wermuth (at) icloud dot com